Performability Analysis of Networks-on-Chips

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart zur Erlangung
der Würde eines Doktors der Naturwissenschaften
(Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Jie Hou

aus Henan/China

Hauptberichter:	Prof. Dr.-Ing. Martin Radetzki
Mitberichter:	Prof. Dr. rer. nat. habil. Ilia Polian

Tag der mündlichen Prüfung: 28.04.2021

Institut für Technische Informatik der Universität Stuttgart

2021

Erklärung

Hiermit versichere ich, diese Dissertation selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, 15.12.2020
Jie Hou

Acknowledgements

I would like to sincerely thank my supervisor Prof. Dr-Ing. Martin Radetzki for allowing me to become a part of his research group, providing me an interesting research direction, his extremely valuable guidance, feedback, and patience during my doctoral research. Besides, I thank Prof. Dr. rer. nat. habil. Ilia Polian for reviewing my thesis.

I also would like to thank my colleagues at the university for their collaboration and support, especially Sabine, Lothar, Helmut, Leandro, and Manuel.

Without the endless love and unconditional support of my entire family members and friends, I will not finish this work. I am very grateful to them. A big thank you to Thomas and Susann, who always encourage me. Especially, a big thank you to my wife and son for their patient tolerance.

Contents

List of Figures xi

List of Tables xv

1 Introduction 1
1.1 Contribution 2
1.2 Thesis outline 3

2 Preliminaries and related work 5
2.1 NoC basics 5
2.1.1 Components 5
2.1.2 Topology 6
2.1.3 Flow control 10
2.1.4 Routing 14
2.2 Performability analysis 19
2.3 Short introduction to stochastic process 20
2.3.1 Discrete-time Markov chain 21
2.3.2 Continuous-time Markov chain 23
2.3.2.1 Kolmogorov equations and generator matrix 25
2.3.2.2 Transient probabilities of CTMCs 27
2.3.2.3 Steady-state probabilities of CTMCs 29
2.3.2.4 An example of CTMC analysis 30
2.4 Related work 33
2.4.1 Performance analysis of NoCs 33
2.4.1.1 Simulator based performance analysis of NoCs 33
2.4.1.2 Performance analysis of NoCs in an analytical way 34
2.4.2 Reliability analysis of NoCs 37
2.4.3 Research of performability analysis 38

3 Markov modeling of NoCs 41
3.1 CTMC model of NoCs 41
3.2 Use case: Markov modeling of 2D mesh-based NoCs under transient faults 45

3.3 Use case: Markov modeling of 2D mesh-based NoCs under permanent faults 48
3.4 Use case: Markov modeling of 3D partially-connected mesh-based NoCs 50
3.5 Tools for Markov model generation and analysis 52
3.5.1 Markov model generator 52
3.5.2 Markov model analysis 55

4 Performance metrics and their computation 57
4.1 Communication time 57
4.1.1 Definition of communication time 58
4.1.1.1 End-to-end communication flow 58
4.1.1.2 Communication round 59
4.1.1.3 Latency of an end-to-end communication flow 60
4.1.1.4 Latency of a communication round 60
4.1.1.5 Communication time 61
4.1.2 Zero-load latency 62
4.1.3 End-to-end communication flow latency with multiple flows in NoC 64
4.1.3.1 Shared channel bandwidth 64
4.1.3.2 Refined computation of shared channel bandwidth 65
4.1.3.3 Estimation of end-to-end communication flow latency with network contention 68
4.1.4 Evaluation of proposed equation and refinement procedure 68
4.1.4.1 Evaluation case: fault-free 69
4.1.4.2 Evaluation case: 10 percent faults 71
4.1.4.3 Evaluation case: 20 percent faults 73
4.2 Fault resilience 75
4.2.1 Powers of adjacency matrix 75
4.2.2 Channel connectivity matrix 77
4.2.2.1 Channel dependency matrix 77
4.2.2.2 Powers of channel dependency matrix 80
4.2.3 The fast approach to computing connectivity 83
4.2.4 Workflow of computing fault resilience 87
4.2.5 Evaluation of our approach to computing fault resilience 88

4.3 Machine learning based approach to computing performance metrics 89
4.3.1 Overview of the machine learning based methodology 91
4.3.2 Datasets generation workflow 92
4.3.3 Model selection 95
4.3.4 Model evaluation 98
4.4 Multi-threading based tool to obtaining performance metrics 99

5 Case studies 103
5.1 Evaluation preliminary 103
5.1.1 Monte Carlo method 103
5.1.2 Normalization of communication time 104
5.2 Performability evaluation of mesh NoCs concerning communication time 105
5.2.1 Experimental setup 106
5.2.2 Analysis of long-term communication time of XY routing 107
5.2.3 Long-term communication time comparison of XY and FTNF routing algorithms 110
5.3 Performability evaluation of mesh NoCs concerning fault resilience 111
5.3.1 Experimental setup 112
5.3.2 Result analysis of long-term fault resilience 113
5.3.3 Result analysis of long-term fault resilience under a high local repair rate 114
5.3.4 Result analysis of transient fault resilience 115
5.4 Machine learning based performability evaluation 116
5.4.1 Long-term communication time under different packet injection rates 116
5.4.2 Long-term fault resilience under different repair rates 118
5.5 Transient performability evaluation of 3D NoCs 119
5.5.1 Implementation of 3D-FTNF routing 119
5.5.2 Analysis of transient communication time 122
5.5.3 Analysis of transient fault resilience 123
5.5.4 Determining maximum failure rates of links and TSVs 124
5.6 Transient performability evaluation of hexagonal NoCs 126
5.6.1 Transient communication time of different sizes of hexagonal NoCs 127

5.6.2 Transient communication time comparison of a hexagonal NoC and a mesh NoC 129

6 Conclusion and future work 131
6.1 Future work 134

Abbreviations 135

References 137

List of Figures

2.1 An example of a NoC-based many-core on-chip system 6
2.2 Commonly used direct NoC topologies: ring, mesh, torus, hexagonal mesh 7
2.3 An example of indirect NoC topology 8
2.4 Examples of 3D meshes 9
2.5 Data units on NoCs: package, flit, and phit 10
2.6 Time-space diagram for an example of store-and-forward flow control 12
2.7 Time-space diagram for an example of wormhole flow control 13
2.8 An example of Valiant's routing on a 2D mesh 15
2.9 Minimal adaptive routing on a 2D mesh 16
2.10 Routing-dependent deadlock on a 2D mesh 16
2.11 Possible turns allowed in the turn model and in XY routing on a 2D mesh 17
2.12 State of a computing core over time 21
2.13 Sample path of a CTMC 24
2.14 Markov model of a multiprocessor system of 4 processors 32

3.1 Router groups of a mesh-based NoC of size 4x4 47
3.2 Markov model of a mesh-based NoC under consideration of transient faults 48
3.3 Markov model of a mesh-based NoC under consideration of permanent faults 49
3.4 An example of a vertically-partially-connected 3D mesh of size 4x4x3 51
3.5 Markov model of a partially-connected 3D mesh of size 4x4x3 52
3.6 Structure of Markov model generator 53
3.7 Structure of Markov model analysis tool 55

4.1 Relationship between communication time, communication round, and communication flow 58
4.2 Example of an end-to-end communication flow on a mesh-based NoC of size 3x3 61

4.3 Example of a communication round on a mesh-based NoC of size 3x3 62
4.4 Time-space diagram of an end-to-end communication flow on a mesh-based NoC of size 3x3 64
4.5 Flows arrive at Router R_4 at the same time 65
4.6 Flows arrive at Router R_5 at different time 66
4.7 Average latency comparison under fault-free condition 70
4.8 Average latency comparison under fault condition (10 percent) 72
4.9 Average latency comparison under fault condition (20 percent) 74
4.10 A 5-node directed graph with 6 edges 76
4.11 Forbidden turns of a mesh-based NoC of size 3x3 under FTNF routing 78
4.12 A mesh-based NoC of size 3x3 with two faulty routers 81
4.13 Example of computing connectivity of a north-east source-destination pair 86
4.14 Example of computing connectivity of a north-west source-destination pair 87
4.15 Overview of the methodology to predict NoC's performance . 92
4.16 Workflow of datasets generation 92
4.17 Global average delays under different packet injection rates for mesh 8x8 with uniform random traffic 95
4.18 Example of 5-fold cross-validation 97
4.19 Structure of our proposed tool to obtain performance metrics in parallel 100

5.1 Aspects of performability evaluation 103
5.2 Transient performabilities of meshes with different sizes 109
5.3 Long-term communication time comparison of XY and FTNF routing algorithms 110
5.4 Average communication times of XY and FTNF routing algorithms on meshes 6×6 and 8×8 111
5.5 Long-term fault resilience comparison for different size meshes 113
5.6 Long-term fault resilience comparison with local repair rate of 0.04 h^{-1} 114
5.7 Transient fault resiliences on the mesh NoC of size 10×10 . . 115
5.8 Long-term communication times under different packet injection rates 117
5.9 Long-term fault resiliences under different repair rates 118

5.10 Transient communication times of 3D-FTNF and elevator-first routing algorithms 124
5.11 Transient fault resiliences of 3D-FTNF and elevator-first routing algorithms 125
5.12 Markov model for both mesh and hexagonal NoCs under consideration of permanent router faults 127
5.13 Transient communication times of three different sizes hexagonal NoCs 128
5.14 Transient communication times of a 10x10 hexagonal NoC and a 10x10 mesh NoC 129

List of Tables

2.1 Fault-tolerant negative-first routing algorithm proposed in [58] 18

3.1 List of symbols for Markov modeling 44
3.2 Fault limit (10 percent) and states information for meshes with different sizes under consideration of transient faults of routers 55

4.1 List of symbols for communication time 59
4.2 Simulation time of full round under fault-free condition 71
4.3 Simulation time of full round under fault condition (10 percent) 73
4.4 Simulation time of full round under fault condition (20 percent) 73
4.5 Average fault resilience for different size meshes 89
4.6 Total execution time for different size meshes 89
4.7 Cross-validation scores of regression models for communication time 97
4.8 Cross-validation scores of regression models for fault resilience 97
4.9 Evaluation scores of models for communication time 98
4.10 Evaluation scores of models for fault resilience 98

5.1 Communication time comparison of different size meshes for successfully delivering 5000 packets 107
5.2 Long-term performabilities and communication times of meshes with different sizes 108
5.3 Long-term residing probabilities in valid and failure states . . . 108
5.4 Break-even failure rates for meshes with different sizes 109
5.5 Fault limit (20 percent) and states information for meshes with different sizes 112
5.6 Determined MFRs for 3D-FTNF and elevator-first routings . . 125
5.7 Base communication times of three different sizes hexagonal NoCs 128

Abstract

The rapidly increasing transistor density enables the evolution of many-core on-chip systems. Networks-on-Chips (NoCs) are the preferred communication infrastructure for such systems. Besides, NoCs have also been proposed to solve the complex on-chip communication problem in the three-dimensional systems-on-chips (3D SoCs). A downside of technology scaling is the increased susceptibility to failures in NoC resources. Ensuring reliable operation despite such failures degrades NoC performance and may even invalidate the performance benefits expected from scaling. Thus, it is not enough to analyze performance and reliability in isolation, as usually done.

The goal of this thesis is to cope with the performance and reliability analysis of NoCs jointly under consideration of faults. This is achieved by the concept of the performability analysis. One of the commonly used performability methods is the Markov reward model. In this work, a generic methodology based on the Markov reward model is proposed to perform performability evaluation and analysis of NoCs under various design parameters. The introduced methodology consists of two parts. In the first part, generic Markov modeling of NoCs with consideration of different fault models is proposed. It can be applied for both 2D and 3D NoCs. As the size of NoCs increases, the size of their Markov state spaces grows as well. To perform the performability evaluation of large size NoCs, we implement tools to generate the Markov state space and to perform the long-term and transient analyses of the generated Markov model.

In the second part, we introduce two performance metrics namely communication time and fault resilience. Communication time is an indicator of the ability to successfully transmit a certain number of packets. Fault resilience is an indicator of how reliable a NoC is in terms of connected paths. As the number of fault combinations of some states in the utilized Markov model is huge, it is a challenging and necessary research task to obtain performance metrics of valid states in the utilized Markov model efficiently. In this work, we present different approaches to computing the mentioned performance metrics. These approaches have been verified with simulations concerning accuracy and speedup.

Finally, we utilize several case studies to demonstrate how to use our proposed methodology to evaluate and analyze the performability of NoCs.

Performabilities of various topologies and routing algorithms are evaluated and compared with respect to two performance metrics. Moreover, we investigate how performability develops with scaling towards larger NoCs and explore the limits of scaling by determining the break-even failure rates under which scaling can achieve net performance increase.

Kurzfassung

Die schnell ansteigende Transistordichte ermöglicht die Entwicklung von On-Chip-Systemen mit vielen Kernen. Networks-on-Chips (NoCs) sind die bevorzugte Kommunikationsinfrastruktur für solche Systeme. Außerdem wurden NoCs in Erwägung gezogen, um das komplexe Kommunikationsproblem in den dreidimensionalen Systems-on-Chips (3D-SoCs) zu lösen. Ein Nachteil der Technologieskalierung ist die erhöhte Anfälligkeit für Fehler in NoC-Ressourcen. Die Gewährleistung eines zuverlässigen Betriebs trotz solcher Fehler beeinträchtigt die NoC-Leistung und kann sogar die von der Skalierung erwarteten Leistungsvorteile ungültig machen. Daher reicht es nicht aus, Leistung und Zuverlässigkeit wie üblich separat zu analysieren.

Ziel dieser Arbeit ist es, die Leistungs- und Zuverlässigkeitsanalyse von NoCs miteinander unter Berücksichtigung von Fehlern zu bewältigen. Dies wird durch das Konzept der Performabilitätsanalyse erreicht. Eine der am häufigsten verwendeten Performabilitätsmethoden ist das Markov-Reward-Model. In dieser Arbeit wird eine generische Methodik vorgeschlagen, die auf dem Markov-Reward-Model basiert, um die Performability-Bewertung von NoCs unter verschiedenen Entwurfsparametern durchzuführen. Die eingeführte Methodik besteht aus zwei Teilen. Im ersten Teil wird eine generische Markov-Modellierung von NoCs unter Berücksichtigung verschiedener Fehlermodelle vorgeschlagen. Diese kann sowohl für 2D- als auch für 3D-NoCs angewendet werden. Mit zunehmender Größe von NoCs wächst auch die Größe ihrer Markov-Zustandsräume. Um die Performabilitätsbewertung großer NoCs durchzuführen, werden Tools zur Generierung des Markov-Zustandsraums und zur Durchführung der Langzeit- und Transientenanalysen des generierten Markov-Modells implementiert.

Im zweiten Teil werden zwei Leistungsmetriken vorgestellt, nämlich Kommunikationszeit und Fehlerresilienz. Die Kommunikationszeit ist ein Indikator für die Fähigkeit, eine bestimmte Anzahl von Paketen erfolgreich zu übertragen. Die Fehlerresilienz ist ein Indikator dafür, wie zuverlässig ein NoC in Bezug auf verbundene Pfade ist. Da die Anzahl der Fehlerkombinationen einiger Zustände im verwendeten Markov-Modell sehr groß ist, ist es eine herausfordernde und notwendige Forschungsaufgabe, Leistungsmetriken gültiger Zustände im verwendeten Markov-Modell effizient zu erhalten. In dieser Arbeit werden verschiedene Ansätze zur Berechnung der genannten

Leistungsmetriken vorgestellt. Diese Ansätze wurden mit Simulationen hinsichtlich Genauigkeit und Ausführungszeit verifiziert.

Schließlich werden mehrere Fallstudien verwendet, um zu zeigen, wie unsere vorgeschlagene Methodik zur Bewertung und Analyse der Leistungsfähigkeit von NoCs eingesetzt werden kann. Die Leistung verschiedener Topologien und Routing-Algorithmen wird bewertet und in Bezug auf zwei Leistungsmetriken verglichen. Darüber hinaus wird untersucht, wie sich die Leistungsfähigkeit bei der Skalierung in Richtung größerer NoCs entwickelt, auch im Hinblick auf die Grenzen der Skalierung, indem die Break-Even-Fehlerraten bestimmt werden, unter denen die Skalierung eine Steigerung der Nettoleistung erzielen kann.

Chapter 1
Introduction

The transistor density of chips has rapidly increased over the last decades as a result of the ongoing development of technology scaling. Consequently, processor manufacturers integrate more and more processor cores on a single chip. E.g. Intel Teraflops research chip consists of 80 cores [144]. The Pollack's Rule confirmed the trend towards many-core processors. As the feature size becomes smaller and smaller, the future many-core processors are expected to be integrated with hundreds or even thousands of computing cores [19]. A large number of computing cores brings a huge advantage concerning computational power. However, a large amount of data may be exchanged between these computing cores, which imposes a new challenge on the existing bus-based communication infrastructure, because the bus cannot simply supply the necessary bandwidth. Furthermore, the large number of integrated cores cannot communicate with each other in a case that the utilized bus suffers from failure. In other words, the bus fault can result in failure of the whole many-core processor system.

To overcome the mentioned limitations, Network-on-Chip has been introduced as the communication infrastructure for such many-core on-chip systems [16, 32]. NoCs are capable of providing a scalable and parallel communication infrastructure. To design a NoC of good performance, some factors such as topologies, routing algorithms, flow control, etc. need to be considered [76]. Furthermore, NoCs have also been proposed to deal with the complicated communication problems for 3D SoCs, which are implemented by means of stacking dies and connecting them with Through-Silicon-Vias (TSVs) [51, 2].

As technology scaling develops, the transistor size continues to decrease. On the one hand, it enables the evolution of larger size NoCs. On the other hand, it increases the susceptibility of NoCs components such as routers, links or TSVs to suffer from faults [117, 4, 44]. Regarding their duration, these faults are classified into three types: permanent, transient, and intermittent faults. A permanent fault will not disappear once it occurs. A transient fault causes a failure in a component for some time, and the appeared failure disappears after that time. After that, the component works completely well. Comparing to transient faults, an intermittent fault never entirely goes away, and it oscillates between being quiescent and active [83]. However, a NoC can

still operate with degraded performance in the presence of faults. Therefore, such a NoC is classified as a fault-tolerant system. Performance analysis of such systems requires a unified measure of performance and reliability. However, the performance and reliability analysis of NoCs are normally separated. E.g. authors in [109, 18, 67, 136, 154] investigated the performance of NoCs in the simulator-based way. The works in [31, 35, 101] studied the reliability of NoCs in the analytical approach.

Fault-tolerant systems that operate with faults and repairs can be treated as a Markov model [140]. Performability of such systems can be evaluated by means of Markov reward model [141], which is the well-known framework for unifying the performance and reliability of a fault-tolerant system. In another word, performance and reliability of a NoC are evaluated jointly. In this context, reliability is the probability that the evaluated NoC has been operational during a specified time interval. In this work, we propose a methodology based on Markov reward model to evaluate performability of NoCs from various design aspects.

1.1 Contribution

This thesis investigates the unified performance and reliability of NoCs under consideration of different fault models. The main contributions are summarized based on our publications [70, 69, 71] and listed as follows:

1. Apply Markov reward model to analyze the performability of NoCs. Propose how to model NoCs as a continuous-time Markov chain under various fault models in a generic manner. To evaluate the performability of large-size NoCs, we implement tools to generate Markov state space of NoCs and compute long-term steady-state and transient probabilities of the generated Markov models.
2. Propose and verify efficient approaches to computing two performance metrics namely communication time and fault resilience for valid Markov states. These approaches include an analytical approach to computing communication time, a mixed approach to calculating fault resilience, an approach based on machine learning techniques to computing both communication time and fault resilience, and an approach to obtain performance metrics with parallel processing.

3. Evaluate and analyze the performability of NoCs from different aspects such as topologies and routing algorithms. Several case studies covering both 2D and 3D NoCs are presented and discussed.

1.2 Thesis outline

This thesis is organized as follows:

- In chapter 2, we introduce the general design aspects of NoCs such as topology, flow control, routing. Besides, the concept of performability and the performability framework named Markov reward model are discussed. Finally, an overview of related work covering performance, reliability, and performability from the NoC research field is given.
- Chapter 3 is about how to model NoCs as a continuous-time Markov chain. First, we present a generic Markov model for both 2D and 3D NoCs under consideration of different fault models. This model consists of two parts: generic Markov states and transitions between these states. Then, we apply our proposed Markov model to three different use cases covering both 2D and 3D NoCs. Besides, we develop tools to generate the Markov state space of NoCs and to perform long-term and transient analyses of the generated Markov models.
- Chapter 4 targets at performance metrics (namely communication time and fault resilience) and their computation. First, an analytical approach to computing communication time is presented. Then, we present how to compute fault resilience based on the channel dependency matrix and search. Moreover, machine learning and multi-threading based approaches enable us to obtain performance metrics under consideration of more practical parameters such as injection rates. The proposed approaches are verified and compared with simulations.
- In chapter 5, we evaluate and analyze the performability of NoCs based on our proposed Markov modeling and performance metrics. Different evaluation case studies for various NoC architectures and fault models are investigated. Based on the experimental results, we can conclude that our proposed performability evaluation methodology is very general and highly efficient.
- Chapter 6 concludes the thesis and provides an outlook on future work.

Chapter 2
Preliminaries and related work

2.1 NoC basics

With the development of integration technology, transistor density has been increased drastically. It helps microprocessors move from single-core to many-core architectures. The traditional bus-based communication infrastructure is not suitable for such large many-core systems, as it cannot provide the required area, power, and latency requirements. Inspired by the techniques of internet and high-performance-computing systems, different research groups proposed a new communication architecture named Network-on-Chip for many-core on-chip systems [16, 32, 60]. NoC is a highly scalable, bandwidth-efficient, and usually packet-switched network, which allows the parallel communication between cores. In general, the design space of NoCs is larger comparing to bus-based solutions, as different topologies, routing algorithms, and flow control mechanisms could be implemented [76]. Moreover, NoCs can tolerate faults and cope with communication bottlenecks due to their inherent redundancy provided by multiple alternative paths in the network.

2.1.1 Components

A NoC is composed of three main components: links, routers, and network interfaces (NIs). Figure 2.1 illustrates an example of a NoC-based many-core on-chip system. Links are used to connect components in the network. In this thesis, a link consists of two unidirectional communication channels enabling full-duplex communication between the connected components.

The network interface establishes the logical connection between the processing elements (PE) and the network. It can be viewed as a wrapper converting the I/O protocol of the processing elements into the protocol used within the NoC. Processing elements can be treated as intellectual property (IP) cores such as processors, digital signal processors (DSPs), and so on. In this work, PEs are the sources and destinations of communication pairs.

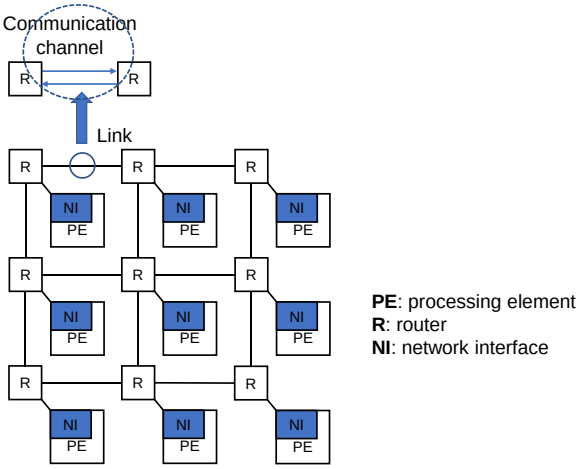


Fig. 2.1: An example of a NoC-based many-core on-chip system

Commonly, messages transmitted on NoCs are packet-based. NIs are in charge of packetization and depacketization of data. A router is a core component of a NoC. It is in charge of transmitting packets. Generally, it consists of input and output ports, buffers for storing incoming packets, and a switching unit. The functionality of the switching unit is to forward packets based on the implemented routing algorithms.

2.1.2 Topology

A NoC's physical layout is defined by its topology, which specifies connections between routers and links. Mathematically, a topology can be represented by a graph containing a set of routers and a set of links. In the research field of NoCs, there are two commonly used topology categories: direct topologies and indirect topologies [29]. In the direct topologies, every router is connected to a PE and vice versa. Such a pair is generally called a node. The commonly used direct topologies are ring, mesh, torus, and hexagonal mesh. In the indirect topology, not all routers are associated with a processing element.

Instead, some routers are only responsible for forwarding packets through the network. E.g. a 3-stage butterfly illustrated in Figure 2.3 is an example of the indirect topology. In general, we characterize a topology using the following

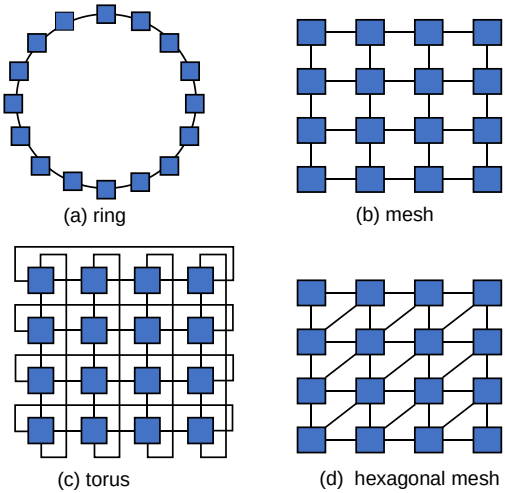


Fig. 2.2: Commonly used direct NoC topologies: ring, mesh, torus, hexagonal mesh

parameters:

- Degree: the number of links connected to a router is its degree. In case that all routers of a NoC have the same degree, such a NoC is called a regular one. Otherwise, a NoC is categorized as an irregular one. A larger number of links can increase throughput. However, a large number of links results in higher costs and could make the layout of a NoC more complex. As can be seen, the ring and torus NoCs are regular, as routers have the same degree. E.g. routers of a torus NoC have a degree of 4. Each router is associated with 4 links. In other words, each router has 4 neighbors. Comparing to ring or torus, the mesh or hexagonal mesh topologies do not have the same degree. On a mesh NoC, routers located at corners have a degree of 2, routers on the four edges have a degree of 3, and inner routers have a degree of 4. In chapter 3, we group routers of a mesh NoC based

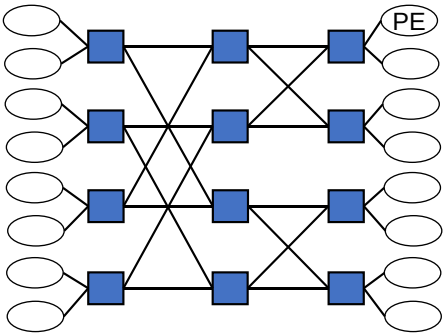


Fig. 2.3: An example of indirect NoC topology

on their degrees when we model it as a continuous-time Markov chain. Similarly, routers of a hexagonal mesh NoC have degrees of 2, 3, 4, and 6 respectively.

- Hop count: hop count gives the information on the number of links a packet takes from its source to its destination.
- Diameter: the maximum shortest path between all pairs of routers of a NoC is defined as its diameter. Generally, the diameter is an important metric as the packet delay is proportional to the number of hops [25]. E.g. the diameter of a mesh or a hexagonal mesh of size $n \times n$ is 2(n-1). In contrast, a torus of size $n \times n$ has the diameter of (n-1).
- Bisection width: bisection width provides the number of links that need to be removed to divide a NoC into two sub-networks of equal size. If a NoC has a larger bisection width, it implies that more paths exist between two sub-networks. In another word, the blocking probability inside such a NoC is lower. Therefore, a large bisection width is preferred.

3D Topology

In recent years, 3D integration technology enables a new dimension to exploit novel integration of vertical silicon dies utilizing vertical links. One of the most popular technologies for constructing vertical links is Through-Silicon-Via, which allows fine pitch, high density, and high compatibility with standard

CMOS process [17]. In comparison to 2D integrated circuits, TSVs can provide short interconnects, improve the integrated density of chip, and reduce the on-chip power consumption [139]. By combining the 3D integration technology and the NoC technology, 3D NoC becomes a promising solution to solve the complex on-chip communication issues in the stacked many-core SoCs [51, 2].

In recent years, various architectures have been proposed and studied for 3D NoCs, e.g. 3D mesh, 3D torus, Honeycomb 3D NoC, and so on [118]. Figure 2.4 shows two examples of 3D meshes: a vertically-fully-connected 3D mesh and a vertically-partially-connected 3D mesh. In a vertically-fully-connected 3D mesh, each router on a horizontal layer is associated with two TSVs: upstairs TSV and downstairs TSV. In contrast, only some routers in a vertically-partially-connected 3D mesh have TSVs. Besides, some routers can have either an upstairs TSV or a downstairs TSV.

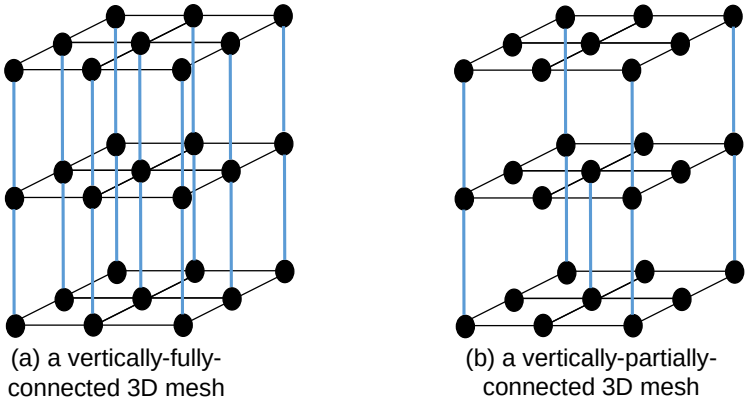


Fig. 2.4: Examples of 3D meshes

Even though 3D integration technology potentially introduces a whole new set of application possibilities, the TSVs also bring their disadvantages. First, as the number of network nodes increases in each layer, the number of TSVs increases as well, which leads to high area consumption. Second, fabricating a 3D integrated circuit using TSV technology requires several extra and expensive manufacturing steps. Each extra step adds a risk for defects

[159]. In this work, we focus on vertically-partially-connected 3D NoC. We assume a full 2D mesh topology for each horizontal layer. The allocation of TSVs is out of the scope of this work.

2.1.3 Flow control

Flow control regulates the allocation of resources (buffers and communication channels) and how they are shared among many messages on a NoC. As mentioned previously, when a message is injected into a NoC, it is first segmented into packets by a network interface. As the widths of buffers and communication channels are normally limited, packets are further divided into fixed-length flow control units (flits). Each packet will consist of a head flit, an arbitrary number of body flits, and a tail flit. The destination information is stored in the head flit. The tail flit denotes the end of a packet. Furthermore, a flit can be subdivided into a certain number of physical information units (phits). A phit is the smallest transfer unit on a NoC [76, 33]. The decomposition of the message into packets, packet into flits, and flit into phits are illustrated in Figure 2.5. In this session, three different flow control techniques will be discussed: circuit switching, packet-based flow control, and flit-based flow control.

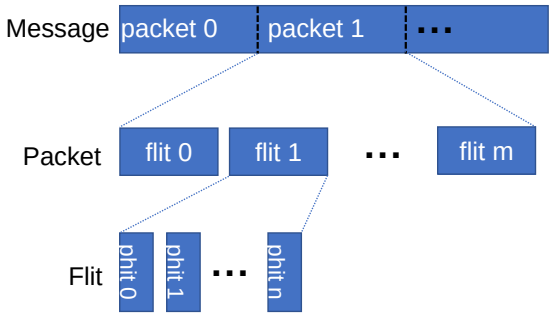


Fig. 2.5: Data units on NoCs: package, flit, and phit

Circuit switching

Circuit switching targets at transmitting large messages. It works as follows:

- A request message is sent from the source router to the destination router. Besides, it reserves the communication channel along with the propagation. During this process, the propagated request message could be blocked at a router if the next channel has already been reserved for other messages.
- After the request message arrives at its destination router, an acknowledgment message will be sent back to the source router.
- Once the source router receives the acknowledgment message from the destination router, the source router can send data messages to the destination using the reserved communication channels without further control.
- After all data messages are sent to the destination router, the source router will send a teardown message to deallocate the reserved communication channels.

In circuit switching, communication channels are reserved in advance before data transmission starts, therefore buffers are not necessary at each router to hold incoming data packets that need allocation. Thus, circuit switching is treated as a buffer-less flow control method. However, the path between the source and the destination needs to be traversed three times, the circuit switching method will result in high latency for small messages. On the other hand, huge overhead and inefficiency because of the connection setup will be introduced in case that communication partners change frequently.

Packet-based flow control

Packet-based flow control is one type of buffered flow control, which adds buffers to routers of a NoC to buffer incoming packets. Buffered flow control enables us to decouple the allocation of the adjacent communication channels in time [76, 33]. E.g. a packet arrives at a buffer of a router on cycle h and stays in that buffer for a number of i cycles until the output communication channel is successfully allocated on cycle $(h+i)$. If no buffering is utilized, the packet arriving on cycle h should level that router on cycle $(h+1)$.

Comparing to circuit switching, packet-based flow control operates at the packet granularity. In another word, the message is partitioned and transmitted as packets. Besides, the first few bytes of a packet store routing information. Thus, each packet can be routed from source to destination separately. One

typical packet-based flow control method is the store-and-forward (SAF) flow control. Under SAF control, each router located on the route of a packet waits until that packet has been completely received and then transmits the packet to the next router. Moreover, the minimum size of buffers of routers is one packet. Figure 2.6 illustrates a time-space diagram of the transmission of a

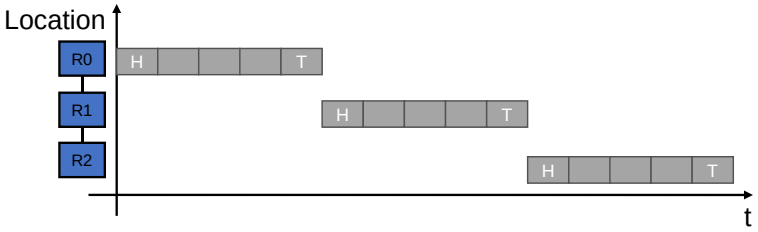


Fig. 2.6: Time-space diagram for an example of store-and-forward flow control

packet under SAF flow control. The packet consists of 5 flits and is being forwarded from router *R0* to router *R1* with no contention. From the figure, we know that the packet can be only forwarded at the next router after it has been completely received. It takes 15 cycles to transmit the packet to router *R2*. It can be concluded that the packet latency with SAF flow control is proportional to the distance between source and destination.

Flit-based flow control

The two major drawbacks of SAF control are large packet latency and large-sized buffers. It is difficult to construct small and compact routers under the need to buffer complete packets. Flit-based flow control can overcome the drawbacks of packet-based control flow. In this control scheme, channel bandwidth and buffers are allocated on the flit granularity. It means that the size of buffers could be smaller than one packet. Besides, flit-based flow control operates at the flit granularity. A packet can be transmitted at the next

router even if it has not been completely received. One typical flit-based flow control method is the wormhole flow control [76]. It works as follows:

- The head flit of a packet can be transmitted to the next router once two resources are available: the desired output communication channel to the next router and the buffer at the corresponding port of the next buffer has at least one-flit available space. After the head flit is forwarded, the utilized communication channel is reserved for that packet. Simply speaking, the functionality of a head flit is to reserve the communication channels along the route from its source to its destination.
- Body flits of that packet use the communication channel reserved by the head flit and require only a flit buffer to advance.
- The purpose of the tail flit is to release the communication channels reserved by the head flit. After that, the communications channels are available to other flits.

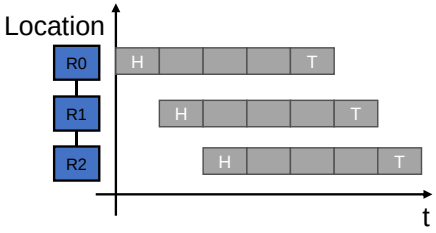


Fig. 2.7: Time-space diagram for an example of wormhole flow control

Figure 2.7 illustrates an example of wormhole flow control. In this example, a 5-flit packet is transmitted from router *R0* to *R2* without contention. We assume that the buffer size is 2 flits. As can be seen, this packet is transmitted in a pipelined manner. The head flit arrives at router *R2* on cycle 3. It only takes 7 cycles to transmit this packet. Comparing to store-and-forward flow control, wormhole flow control needs small-sized flit buffers and reduces packet latency.

2.1.4 Routing

The routing algorithm is an intensive research direction of NoC. Generally, several paths exist between two routers on many topologies. The path chosen by a packet from its source to its destination is determined by the routing algorithm.

Routing classification

Routing algorithms can be mainly categorized into three types: deterministic, oblivious, and adaptive [76, 33].

Under deterministic routing, the same path is always provided for a given source and destination pair. Moreover, deterministic routing does not take network status into account when the route is determined. A typical example of deterministic routing is dimension-ordered routing (DOR), in which coordinates (e.g. x- and y- coordinates for 2D NoCs) are used as router addresses. A packet is normally transmitted first along one coordinate until reaching an intermediate router that has the same corresponding coordinate as the destination, after that the packet is further forwarded along another coordinate. XY routing is a well-known example of DOR routing. With the XY routing, a packet is first forwarded along x-coordinate. Once this packet arrives at an intermediate router which has the same x-coordinate as the destination. After that, the packet propagates along the y-coordinate to its destination.

The target of oblivious routing is to balance the network load. Under oblivious routing, the utilized routing method chooses different intermediate routers along the route from source to destination. Network status such as faults or congestions are not considered in the oblivious routing algorithm. A well-known example of an oblivious routing algorithm is Valiant's algorithm [142]. Under Valiant's routing, an intermediate destination will be randomly selected. After that, the packet is first transmitted to this selected intermediate destination and then routed to its final destination. Figure 2.8 displays an example of Valiant's routing algorithm on a mesh NoC of size 4x4. The black-arrowed path is the route utilized under Valiant's routing. The path illustrated with a dashed line is the route selected by XY routing. As can be seen, Valiant's algorithm uses non-minimal paths to balance network load.

Adaptive routing is also called dynamic routing, which is capable of providing several valid paths for a given source and destination pair under con-

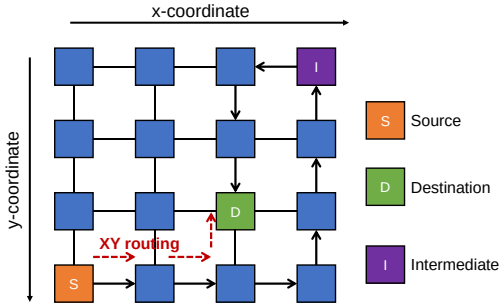


Fig. 2.8: An example of Valiant's routing on a 2D mesh

sideration of network status. Information on buffer occupancy and operation status of a router is available to its neighbors. Adaptive routing can be further categorized as minimal or non-minimal adaptive. With minimal adaptive routing, only routes that have minimum hop count are adaptively chosen while a packet propagates to its destination. Figure 2.9 illustrates an example of a minimal adaptive routing algorithm on a mesh NoC of size 3x3 with consideration of one faulty router. In this displayed example, there is no path between source and destination routers with XY routing. However, two available routes denoted by two dash lines exist for the given source and destination pair under minimal adaptive routing. Comparing to minimal adaptive routing, non-minimal adaptive routing can provide connected paths between all routers as long as they are connected.

The turn model

An important aspect of designing a routing algorithm is to avoid deadlock, which describes a situation in which packets are stuck in the buffers of routers and cannot be transmitted further. Especially, deadlocks are prone to occur on NoCs utilizing the wormhole flow control method [103]. Generally, the reason for causing deadlock is circular dependencies between the communication resources of a NoC. An example of a deadlock situation is displayed in Figure 2.10. Four packets (P1, P2, P3, P4) are involved in this example. Packet P1 needs the communication channel that is reserved by packet P2 to forward.

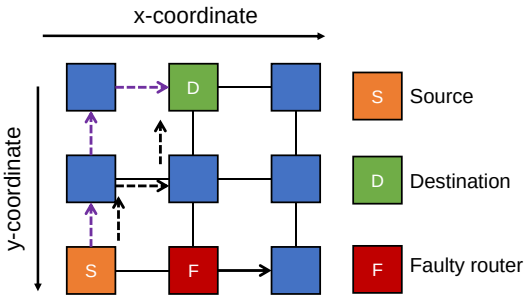


Fig. 2.9: Minimal adaptive routing on a 2D mesh

It means that packet P1 is blocked by packet P2. Similarly, packets P2, P3, and P4 are blocked by packet P3, P4, and P1 respectively. As can be seen, a circular dependency on communication channels is formed in this example.

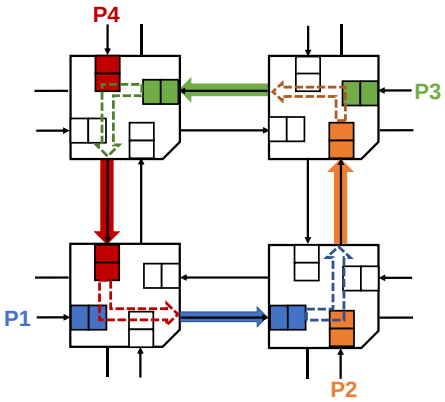


Fig. 2.10: Routing-dependent deadlock on a 2D mesh

The common approach to preventing deadlocks is to restrict some turns that a packet can take. Authors in [57] proposed the well-known turn model

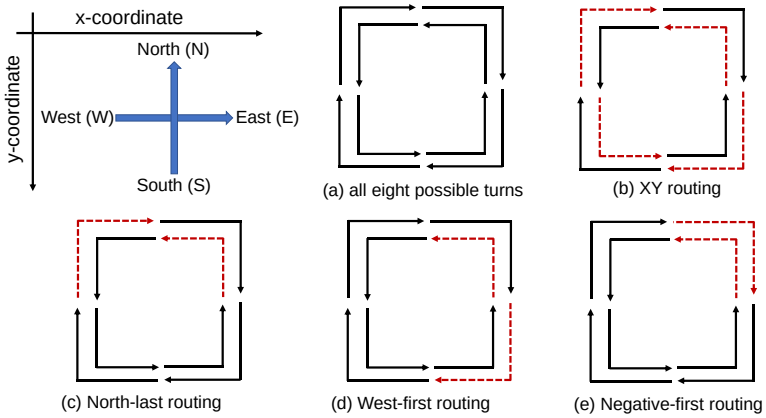


Fig. 2.11: Possible turns allowed in the turn model and in XY routing on a 2D mesh

for mesh-based NoCs. Based on the turn model, a turn is formed when a flit that propagates in a certain direction is forwarded to another direction. There are eight different turns in a 2D mesh: four turns in the clockwise direction and four turns in the anticlockwise direction, which are displayed in Figure 2.11(a). In this dissertation, the direction of the x-coordinate starts from west to east. Similarly, the direction of the y-coordinate is from north to south.

With XY routing, y-to-x turns are forbidden. Once a packet is forwarded to y-direction, it is not allowed to turn in x-direction any more. Thus, there are only two allowed turns both in the clockwise and anticlockwise directions as illustrated in Figure 2.11(b). Therefore, XY routing is inherently deadlock-free.

However, the potential adaptiveness of a routing algorithm is limited in case that more turns than necessary are forbidden. The well-known turn model only prohibits one turn in both clockwise and anticlockwise cycles. The commonly used adaptive routing algorithms based on the turn model are north-last, west-first, and negative-first. Under the north-last routing as displayed in Figure 2.11(c), two turns namely the north-to-west turn and north-to-east turn are prohibited. In other words, turns whose start direction is north should be forbidden. Therefore, packets cannot change direction any more once they are transmitted towards the north direction. In the west-first routing shown

in Figure 2.11(d), the north-to-west and south-to-west turns are forbidden. Thus, a packet must first select the west direction to travel west. For negative-first routing displayed in Figure 2.11(e), all turns that start from a positive direction and end in a negative direction are not allowed. In the 2D mesh topology, west and south are defined as negative directions, north and east denote positive directions. Besides, a fault-tolerant version of the negative-first routing algorithm was introduced by authors in [58]. Table 2.1 describes briefly how the fault-tolerant negative-first (FTNF) routing works. E.g. in case of a destination router is located at NE direction to a source router, if the packet arrives at the south input port of the current router, the output direction can be E or N based on the offsets along the x- and y- directions. If the x-offset is larger than or equal to y-offset, the preferred output direction is E. Otherwise, the preferred output direction is N.

Table 2.1: Fault-tolerant negative-first routing algorithm proposed in [58]

	Incoming direction				
Destination	Local	N	E	S	W
NE	EN^*	ES	N	EN^*	EN^*
E (1 current router on south edge; 2 others)	EN^1 SE^2	SE	–	E	E
N (1 current router on west edge; 2 others)	NE^1 WN^2	–	NE^1 WN^2	N	N
NW (1 current router on south edge; 2 others)	WN^1 WS^2	WS	WS	WN	NE
W (1 current router on south edge; 2 others)	WN^1 WS^2	WS	WN^1 WS^2	WN	–
SE (1 current router on west edge; 2 others)	SE^1 SWE^2	SE^1 SWE^2	SW	EN	SE
S (1 current router on west edge; 2 others)	SE^1 SW^2	SE^1 SW^2	SW	–	SE
SW	WS^*	WS^*	WS^*	WN	SE

$*$: Select the direction that could provide higher path diversity

Virtual channel

A virtual channel is another solution to avoid routing-dependent deadlocks in NoCs. A communication channel can be associated with several virtual

channels. With virtual channels, there is one buffer associated with each input port of a router. With virtual channels, one input port needs to utilize one buffer for each virtual channel. Deadlock is resolved by defining a total order on all the virtual channels and only allowing them to be allocated in either ascending or descending order. Simply speaking, circular communication channel dependencies can be avoided in such an approach [34].

2.2 Performability analysis

Generally, performance is measured separately from reliability. For assessing the performance of a system, the assumption is made that the system is fault-free. For computing reliability, only the event of failure occurrence is taken into account. However, these assumptions only hold for the systems that have two states: fully functional state or failure state [49]. Many real-world systems may have a certain number of reduced operational states in addition to the fully-operation and failure states. Such systems are also called fault-tolerant systems. If a component of a fault-tolerant system fails, the system does not crash immediately. The system enters a degraded state in which it provides service at a reduced quality level [87]. Performance analysis of such systems requires taking the impact of faults and the likelihood of their occurrence into account. This is achieved with the concept of performability [93, 94]. Performability is a unified measure of performance and reliability and can quantify the ability of a system to perform in the presence of faults [124].

Markov reward models are the most commonly used tools for performability analysis. Formally, they contain two parts: a continuous-time Markov chain (CTMC) with state space Ω and a reward function r where r: $\Omega \rightarrow \mathbb{R}$. Each state $i \in \Omega$ is assigned a reward. Generally, the reward associated with a state stands for the performance level given by the system while it is in that state [141].

A CTMC is represented by states and transitions between states. Information about states and transition rates are described by a generator matrix $\mathbf{Q}$. Usually, a CTMC would be used for studying the dynamics of a system. Many techniques and algorithms have been developed to solve long-term steady-state distribution, transient state probabilities, and occupancy times of Markov chains [129]. In our work, the uniformization algorithm is utilized.

As NoC routers are assumed to suffer from transient failures, it is worth knowing how NoC behaves in the long-term perspective. Another question

is its performance at a particular time instance, e.g. 100 hours after starting operation. In order to answer such questions, two definitions named long-term steady-state performability and transient performability are extracted from [141, 131]. Their mathematical expressions are as follows:

- Long-term steady-state performability (Ψ) is defined as:

$$\Psi = \sum_{i \in \Omega} \pi_i r_i \tag{2.1}$$

  where π_i means the long-term steady-state probability of residing in state i. The equation gives the expected reward rate of the system in the long-term perspective.
- Transient performability ($\Gamma(t)$) is expressed as:

$$\Gamma(t) = \sum_{i \in \Omega} \pi_i(t) r_i \tag{2.2}$$

  where $\pi_i(t)$ denotes the probability that the system is in state i at a time instance of t. This equation's meaning is the expected reward rate of the system at a certain point in time.

2.3 Short introduction to stochastic process

Stochastic processes are broadly utilized as mathematical models of systems and phenomena that seem to evolve in a random manner [11]. They have been applied in many technology and engineering fields such as computer science [14], information theory [30], telecommunications [12] and signal processing [40].

Formally, a stochastic process is defined as $\{X_t, t \in T\}$, where the index t usually denotes time and the set T is the index set of the process [52, 39]. Each instance X_t is characterized by a distribution function. There are two widely used types of index sets: discrete-time $T = \{0, 1, 2, \ldots\}$ and continuous-time $T = [0, \infty)$. Discrete-time stochastic processes can be viewed as sequences of random variables, and continuous-time processes are uncountable collections of random variables [39].

Let S be the set of values that X_t can take for any t. Then S is the state space of the stochastic process $\{X_t, t \in T\}$. The value of X_t is called the state

of the evaluated stochastic process at index t. When the value of X_t changes, the process has had a state transition [78]. The random variables in the state space of a stochastic process can be discrete or continuous.

Let us consider a computing core that works for some time, then it suffers from some transient faults. It fails and stays down until it is repaired. After that, it becomes operational again. The described process repeats over time. The behavior of this computing core is illustrated in Figure 2.12. We can define a stochastic process to describe the state of the computing core over time. Let X_t be the state of the computing core at time t. Obviously, $X_t, t \in [0, \infty)$ is a continuous-time stochastic process with discrete state space $\{0, 1\}$.

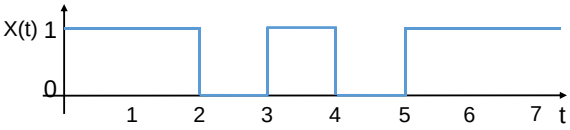


Fig. 2.12: State of a computing core over time

2.3.1 Discrete-time Markov chain

Let us consider a system, which is observed at times of $0, 1, 2, \ldots$. Here, time 0 is the time point at which the evaluated system started its operation. Let X_t be the state of the system at time t. Suppose the current time is $t = 20$. In other words, we have already observed $X_0, X_1, \ldots, X_{20}$. Is it possible that we can predict the state of the system at time $t = 21$? Normally, X_{21} depends on $X_0, X_1, \ldots, X_{20}$. However, a considerable simplification happens if the incoming state X_{21} depends only on X_{20} even given the complete history $X_0, X_1, \ldots, X_{20}$. If the system has such property at all times (not just at $t = 21$), it is a discrete-time Markov chain (DTMC).

Let S be a discrete set. Formally, a DTMC is an indexed collection of random variables $\{X_0, X_1, \ldots\}$ taking values in S with the property that:

$$P(X_{n+1} = j | X_0 = x_0, X_1 = x_1, ..., X_n = i) = P(X_{n+1} = j | X_n = i) \qquad (2.3)$$

where all $x_0, x_1, \ldots, x_{n-1}, i, j \in S$, and $n \geq 0$ [39, 122]. The set S denotes the state space of the DTMC. Equation (2.3) implies the conditional probability on the left-hand side is the same no matter what values $X_0, X_1, \ldots, X_{n-1}$ take. It describes the Markov property in a mathematical manner. In simple terms, this property is described as follows: given the present state X_n of the system , the future state X_{n+1} is independent of its past states $(X_0, X_1, \ldots, X_{n-1})$. The quantity $P(X_{n+1} = j|X_n = i)$ is the one-step transition probability of the DTMC at time n.

A DTMC is time-homogeneous if the probabilities in Equation (2.3) do not depend on n. To be more specific, a time-homogeneous DTMC is:

$$P(X_{n+1} = j|X_n = i) = P(X_1 = j|X_0 = i) \tag{2.4}$$

for all $n \geq 0$ [39, 122]. Equation (2.4) means that the one-step transition probability depends on i and j rather on the actual time instances.

At this point, we point out that we only focus on time-homogeneous Markov chains with a finite state space $S = \{0, 1, 2, \ldots, M\}$. When we mention a DTMC, we always mean time-homogeneous DTMC with finite state space. For such DTMCs, a shorthand notation for the one-step transition probability could be defined as follows:

$$p_{i,j} = P(X_{n+1} = j|X_n = i), \tag{2.5}$$

where $i, j \in S$.

A convenient way to show all the one-step transition probabilities is the matrix form as displayed below:

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,M} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,M} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,0} & p_{M,1} & \cdots & p_{M,M} \end{bmatrix} \tag{2.6}$$

In the matrix P, rows correspond to the starting state and columns represent the ending state of a transition. E.g. the entry in row number 1 and column number M is the probability of going from state 1 to state M in one step.

DTMCs have broad applications. E.g. authors in [158] propose a Markov process model for forecasting the stock market trend. In [61], the authors propose two associated DTMC models and use them together to evaluate per-

formance of a widely used synchronous duty cycling medium access control protocol in wireless sensor networks.

2.3.2 Continuous-time Markov chain

A continuous-time Markov chain models both the transitions between states and the duration of time in each state. Formally, a stochastic process $\{X(t), t \geq 0\}$ with a discrete state space S is called a continuous-time Markov chain if, for all i, j and $x_u \in S$,

$$P(X(s+t) = j | X(s) = i, X(u) = x_u, 0 \leq u < s) = P(X(s+t) = j | X(s) = i) \tag{2.7}$$

where $t, s, u \geq 0$ [39, 122]. Equation (2.7) defines the Markov property in the continuous time. The future state $X(t+s)$ is independent of the past state $X(u)$ given the present state $X(s)$.

The CTMC $\{X(t), t \geq 0\}$ is said to be time-homogeneous if, for $t, s \geq 0$ [39, 122],

$$P(X(s+t) = j | X(s) = i) = P(X(t) = j | X(0) = i) \tag{2.8}$$

It is worth pointing out that we only focus on time-homogeneous CTMC with finite state space $S = \{0, 1, 2, \ldots, M\}$ in this dissertation. For such CTMCs, the transition probability function $p_{i,j}(t), i, j \in S$ is defined as follows:

$$p_{i,j}(t) = P(X(t) = j | X(0) = i) \tag{2.9}$$

Equation (2.9) can be comprehended as the probability that the process is in state j at time t provided that it was in state i at time 0. Obviously, the transition probability function has the following properties:

$$p_{i,j}(0) = \begin{cases} 0, & \text{if i} \neq \text{j} \\ 1 & \text{if i} = \text{j} \end{cases} \tag{2.10}$$

Analogous to DTMCs, the transition probability matrix of a CTMC is defined as follows:

$$P(t) = \begin{bmatrix} p_{0,0}(t) & p_{0,1}(t) & \dots & p_{0,M}(t) \\ p_{1,0}(t) & p_{1,1}(t) & \dots & p_{1,M}(t) \\ p_{2,0}(t) & p_{2,1}(t) & \dots & p_{2,M}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,0}(t) & p_{M,1}(t) & \dots & p_{M,M}(t) \end{bmatrix} \tag{2.11}$$

Note that $P(t)$ is a matrix of functions of time t.

The Markov property is a form of memorylessness, which leads to the exponential distribution. In a CTMC, once a state i is entered, the process stays in that state for an exponentially distributed length $Exp(r_i)$ of time, which is called the transition time T_i. At the end of the transition time in state i, the process makes a sudden transition to state j ($j \neq i$) with embedded transition probability $p_{i,j}$. Once the process goes into state j, it stays there for an $Exp(r_j)$ amount of time and then moves to a new state k ($k \neq j$) with probability $p_{j,k}$. The described sample path of the CTMC is illustrated in Figure 2.13.

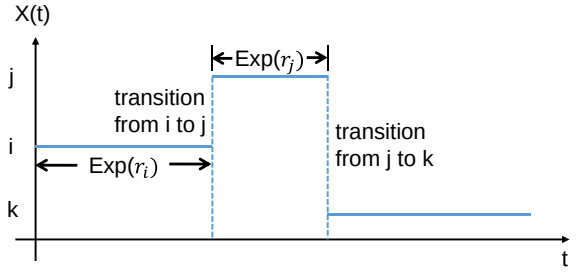


Fig. 2.13: Sample path of a CTMC

Until now, we have seen two symbols that are very similar: $p_{i,j}(t)$ and $p_{i,j}$. Do not confuse $p_{i,j}$ with $p_{i,j}(t)$. Three additional remarks about embedded transition probability $p_{i,j}$ are listed as follows:

1. $p_{i,j}$ is the the probability that the system moves to state j when it leaves state i.

2. $p_{i,i}$ is equal to zero. By definition, the transition time T_i is the time that a CTMC stays in state i until it moves into any other state. Therefore, a transition from state i to itself is not permissible.
3. for any state i from a CTMC, $\sum_{j=0}^{M} p_{i,j} = 1$.

A CTMC contains two parts: embedded transition probabilities P and state rates. States of a CTMC evolve as a DTMC with transition probabilities P. State transitions occur at exponential intervals. Expected value of transition time for state i is equal to $1/r_i$, where r_i can be viewed as the rate of transition out of state i. The transition rate from state i to state j is denoted by $r_{i,j}$.

$$r_{i,j} = r_i p_{i,j} \tag{2.12}$$

Additionally, $r_{i,i}$ is always equal to 0, as the transition to itself is not allowed. The relations between $\{r_i, i \in S\}$ and $\{p_{i,j}, i, j \in S\}$ are given as follows:

$$\begin{aligned} r_i &= \sum_{j=0}^{M} r_{i,j} \\ p_{i,j} &= \frac{r_{i,j}}{r_i} \end{aligned} \tag{2.13}$$

if $r_i \neq 0$. In case that r_i is equal to zero, it means the evaluated CTMC is going to stay in state i forever. $p_{i,j}$ has no meaning at all in such case.

2.3.2.1 Kolmogorov equations and generator matrix

In the area of CTMCs, there are two well-known, proved and widely used equations: Kolmogorov forward and backward equations. They are mathematically described in such a way [39, 122]:

$$p'_{i,j}(t) = -p_{i,j}(t) r_j + \sum_{k \neq j} p_{i,k}(t) r_{k,j} \tag{2.14a}$$

$$p'_{i,j}(t) = -r_i p_{i,j}(t) + \sum_{k \neq i} r_{i,k} p_{k,j}(t) \tag{2.14b}$$

where $p'_{i,j}(t) = \frac{dp_{i,j}(t)}{dt}$. Equation (2.14a) is called Kolmogorov forward equation. And, Kolmogorov backward equation is expressed in Equation (2.14b).

It is convenient to express Kolmogorov forward and backward equations of a CTMC with a finite state space $S = \{0, 1, 2, \ldots, M\}$ in the matrix form:

$$P'(t) = \begin{bmatrix} p'_{0,0}(t) & p'_{0,1}(t) & \ldots & p'_{0,M}(t) \\ p'_{1,0}(t) & p'_{1,1}(t) & \ldots & p'_{1,M}(t) \\ p'_{2,0}(t) & p'_{2,1}(t) & \ldots & p'_{2,M}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p'_{M,0}(t) & p'_{M,1}(t) & \ldots & p'_{M,M}(t) \end{bmatrix} = \begin{bmatrix} p_{0,0}(t) & p_{0,1}(t) & \ldots & p_{0,M}(t) \\ p_{1,0}(t) & p_{1,1}(t) & \ldots & p_{1,M}(t) \\ p_{2,0}(t) & p_{2,1}(t) & \ldots & p_{2,M}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,0}(t) & p_{M,1}(t) & \ldots & p_{M,M}(t) \end{bmatrix} \begin{bmatrix} -r_0 & r_{0,1} & \ldots & r_{0,M} \\ r_{1,0} & -r_1 & \ldots & r_{1,M} \\ r_{2,0} & r_{2,1} & \ldots & r_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M,0} & r_{M,1} & \ldots & -r_M \end{bmatrix} \tag{2.15}$$

$$P'(t) = \begin{bmatrix} p'_{0,0}(t) & p'_{0,1}(t) & \ldots & p'_{0,M}(t) \\ p'_{1,0}(t) & p'_{1,1}(t) & \ldots & p'_{1,M}(t) \\ p'_{2,0}(t) & p'_{2,1}(t) & \ldots & p'_{2,M}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p'_{M,0}(t) & p'_{M,1}(t) & \ldots & p'_{M,M}(t) \end{bmatrix} = \begin{bmatrix} -r_0 & r_{0,1} & \ldots & r_{0,M} \\ r_{1,0} & -r_1 & \ldots & r_{1,M} \\ r_{2,0} & r_{2,1} & \ldots & r_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M,0} & r_{M,1} & \ldots & -r_M \end{bmatrix} \begin{bmatrix} p_{0,0}(t) & p_{0,1}(t) & \ldots & p_{0,M}(t) \\ p_{1,0}(t) & p_{1,1}(t) & \ldots & p_{1,M}(t) \\ p_{2,0}(t) & p_{2,1}(t) & \ldots & p_{2,M}(t) \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,0}(t) & p_{M,1}(t) & \ldots & p_{M,M}(t) \end{bmatrix} \tag{2.16}$$

As can be seen, the rate matrix in Equation (2.15) and Equation (2.16) is same. It is called the generator matrix Q.

$$Q = \begin{bmatrix} -r_0 & r_{0,1} & \ldots & r_{0,M} \\ r_{1,0} & -r_1 & \ldots & r_{1,M} \\ r_{2,0} & r_{2,1} & \ldots & r_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M,0} & r_{M,1} & \ldots & -r_M \end{bmatrix} \tag{2.17}$$

The diagonal elements of the generator matrix Q are non-positive. The values of its elements are defined as follows:

$$q_{i,j} = \begin{cases} r_{i,j}, & \text{if i} \neq \text{j} \\ -r_i & \text{if i} = \text{j} \end{cases} \tag{2.18}$$

It is worth mentioning that the sum of each row in the generator matrix is equal to zero.

Ultimately, the matrix forms of Kolmogorov forward and backward equations are expressed clearly with help of the generator matrix Q:

$$P'(t) = P(t)Q \tag{2.19a}$$

$$P'(t) = QP(t) \tag{2.19b}$$

2.3.2.2 Transient probabilities of CTMCs

Let A be a $m \times m$ matrix. The matrix exponential e^A is the $m \times m$ matrix [65]:

$$e^A = \sum_{n=0}^{\infty} \frac{1}{n!} A^n = \mathbf{I} + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \dots \tag{2.20}$$

In Equation (2.20), $\mathbf{I}$ represents the identity matrix. The matrix version of the exponential function is denoted by the matrix exponential. In case that A is a 1×1 matrix, it reduces the ordinary exponential function. The matrix exponential satisfies many properties of the exponential function. They are as follows:

1. $e^A e^{-A} = e^{-A} e^A = \mathbf{I}$
2. $e^{(s+t)A} = e^{sA} e^{tA}$
3. $\frac{\mathrm{d}e^{tA}}{\mathrm{d}t} = Ae^{tA} = e^{tA}A$

For a CTMC with the generator matrix Q, the matrix exponential e^{tQ} is the unique solution to the forward and backward Chapman-Kolmogorov equations [39].

$$P(t) = e^{tQ} = \sum_{n=0}^{\infty} \frac{1}{n!}(tQ)^n = \mathbf{I} + tQ + \frac{1}{2}(tQ)^2 + \frac{1}{6}(tQ)^3 + \dots$$

$$P'(t) = \frac{\mathrm{d}P(t)}{\mathrm{d}t} = \frac{\mathrm{d}e^{tQ}}{\mathrm{d}t} = Qe^{tQ} = QP(t) \tag{2.21}$$

$$P'(t) = \frac{\mathrm{d}P(t)}{\mathrm{d}t} = \frac{\mathrm{d}e^{tQ}}{\mathrm{d}t} = e^{tQ}Q = P(t)Q$$

Let $\mathbf{p}(t) = [p_0(t), p_1(t), p_2(t), \dots, p_M(t)]$ represent the transient probability distribution of all states at time t, it can be computed using Equation (2.22) given the initial distribution $\mathbf{p}(0)$:

$$\mathbf{p}(t) = \mathbf{p}(0)P(t) = \mathbf{p}(0)e^{tQ} \tag{2.22}$$

Therefore, to compute the transient probabilities, the prerequisite is to determine the initial distribution $\mathbf{p}(0)$ and the generator matrix Q. Generally, it is a numerically challenging task to compute the matrix exponential. The commonly used method is the Padé approximation combined with the scaling and squaring method [9, 68].

Alternatively, uniformization is another commonly used method to compute the transition probability matrix $P(t)$ of a finite state CTMC. Moreover, the uniformization method performs better than the Padé approximation method in most cases [112]. In this method, $P(t)$ is computed as follows [120, 112]:

$$\begin{aligned} \hat{T} &= I + \frac{Q}{r} \\ P(t) &= \sum_{k=0}^{\infty} e^{-rt} \frac{(rt)^k}{k!} \hat{T}^k \end{aligned} \tag{2.23}$$

where $r = \max_{0 \le i \le M} \{|q_{i,i}|\}$, I is the corresponding identity matrix, and $\hat{T}$ is a matrix of the same size of Q.

The term $e^{-rt}\frac{(rt)^k}{k!}$ in Equation (2.23) is the distribution of a Poisson process with rate r. For applied problems, we approximate $P(t)$ by truncating the first N terms of the infinite sum in Equation (2.23). Thus, the error bound is defined as follows:

$$P(t)^N = \sum_{k=0}^{N} e^{-rt} \frac{(rt)^k}{k!} \hat{T}^k$$
$$\left| p_{i,j}(t) - p_{i,j}^N(t) \right| \leq \sum_{k=N+1}^{\infty} e^{-rt} \frac{(rt)^k}{k!} = 1 - \sum_{k=0}^{N} e^{-rt} \frac{(rt)^k}{k!} \tag{2.24}$$

where i,j $\in \{0, 1, 2, \ldots, M\}$.

Suppose ϵ denote the maximum error allowed for any element of $P(t)$, it means that the error bound should not exceed ϵ. In another word, we should choose N that satisfies Equation (2.25):

$$1 - \sum_{k=0}^{N} e^{-rt} \frac{(rt)^k}{k!} \leq \epsilon \tag{2.25}$$

Once N is determined, we can approximate $P(t)$ with the specified accuracy of $(1 - \epsilon)$.

2.3.2.3 Steady-state probabilities of CTMCs

If there are times t_1 and t_2 such that $p_{i,j}(t_1) > 0$ and $p_{j,i}(t_2) > 0$, such a pair of states i and j is called to communicate. A class of a CTMC are states that can communicate with each other. Several classes may exist in a CTMC. If a CTMC has only one class, we call it an irreducible CTMC. The transition probability function of an irreducible CTMC has following properties [39, 122]:

1. $p_{i,j}(t) > 0$, for all $t > 0$ and all states i and j
2. $\lim_{t \to \infty} p_{i,j}(t) = \pi_j$

π_j is called the limiting probability of being in state j and is independent of the initial state i.

For irreducible CTMCs, limiting probabilities are generally referred to as the steady-state probabilities. The steady-state probability π_j of any state j satisfies Equation (2.26).

$$\pi_j = \sum_{i=0}^{M} \pi_i p_{i,j}(t), \text{ for i,j} \in \{0, 1, 2, \ldots, M\} \text{ and t} > 0 \tag{2.26}$$

Furthermore, it has already been proved that the steady-state probabilities of irreducible CTMCs can be solved using the following two equations:

$$\pi_j r_j = \sum_{i \neq j} \pi_i q_{i,j}, \text{ for i,j} \in \{0, 1, 2, \ldots, M\} \tag{2.27a}$$

$$\sum_{i=0}^{M} \pi_i = 1 \tag{2.27b}$$

Let $\boldsymbol{\pi} = \{\pi_0, \pi_1, \pi_2, \ldots, \pi_M\}$ denote the row vector of steady-state probabilities of all states of an irreducible CTMC with state space $S = \{0, 1, 2, \ldots, M\}$, Equation (2.27a) can be written in the matrix form:

$$\pi Q = \mathbf{0} \tag{2.28}$$

Here, $\mathbf{0}$ is a zero vector that has the same size of $\boldsymbol{\pi}$. In addition, Equation (2.27a) provides an intuitive explanation. π_j is the steady-state probability that the evaluated CTMC is in state j and q_j denotes the transition rate out of state j. Therefore, the expression $\pi_j q_j$ at its left-hand side is the rate at which the CTMC goes away from state j. Similarly, as $q_{i,j}$ is the transition rate from state i to state j given that the current process is in state i, each term $\pi_i q_{i,j}$ on its right-hand side is the rate at which the CTMC enters state j. Through summing over all states except from the state j, the whole right-side expression represents the rate at which the CTMC moves into state j from any other state in its state space. Briefly speaking, the overall equation expresses that the rate at which the CTMC leaves state j is the same as the rate that the CTMC enters state j.

2.3.2.4 An example of CTMC analysis

Researchers have applied CTMCs to their works from a wide range of fields such as finance [128], medicine [89, 5, 77, 95], chemistry [7, 24], system biology [62] and computer science [155, 97, 79, 107, 86, 160, 72, 23, 134, 133]. E.g. authors in [7] modeled a reaction network as a CTMC. A reaction network is a chemical system that contains some reactions and chemical species. In their suggested CTMC, the state is the number of molecules of each species and the reactions are the transitions between states. In [128], authors consider a financial market that contains one bank account with stochastic interest rates

and multiple stocks. They assume that the researched market is driven by a geometric Brownian motion with a constant volatility matrix and rates of return modeled as a CTMC. In medical research, [89] presented how to apply CTMC to the transtheoretical model, which is a psychosocial model widely used in studies of health-related outcomes. In the area of computer science, there are many applications in the subareas of performance, reliability, and control. Authors in [72] propose a sequential analytical approach based on CTMC for composite generation and reliability assessment of transmission systems. They construct a CTMC model for the evaluated composite system. Several reliability indexes such as expectation, variance, frequency and duration can be obtained based on their proposed sequential analyses.

In this section, we will provide an example of CTMC. Consider a multiprocessor system that has 4 processors. The processors are identical and the lifetimes of the processors are independent exponential random variables with a failure rate of λ. When a processor is failed, we assume that it can be repaired by some repair process. The repair time processor needs is an exponential random variable with a repair rate of μ. Besides, a repaired processor behaves the same as a functioning one. Let $X(t)$ denote the number of processors that work well at time t. As the failure and repair time of a process follow the exponential distribution, the described multiprocessor system can be modeled as a CTMC.

To model this multiprocessor system as a CTMC, the first step is to define its state space $S = \{4, 3, 2, 1, 0\}$, where the value represents the number of functioning processors. In the state S_0, all 4 processors work well. One of them suffers from a fault, the system moves from state S_0 to state S_1 with a rate of 4λ. In the state S_1, there are 3 operational processors and 1 faulty processor. Therefore, the state S_1 can move back to state S_0 with a rate of μ. Also, the failure of any of 3 operational processors will trigger a transition to state S_2. Similarly, there are 2 functioning processors and 2 faulty processors. The state S_2 can move back to the state S_1 with a rate of 2μ and forward to the state S_3 with a rate of 2λ. In addition, the forward transition rate from state S_3 to S_4 is λ. The backward transition rate of the state S_3 to the state S_2 is 3μ. Finally, in the state 4, all processors are not operational. As each processor has its repair process, any of the 4 faulty processors under repair will move the system to state 1 with a rate of 4μ. The state transition diagram of the system's Markov model is illustrated in Figure 2.14. Its corresponding generator matrix is provided as follows:

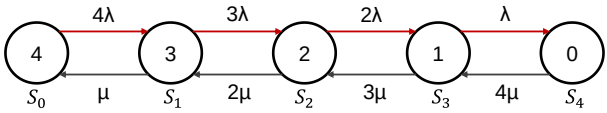


Fig. 2.14: Markov model of a multiprocessor system of 4 processors

$$Q = \begin{bmatrix} -4\lambda & 4\lambda & 0 & 0 & 0 \\ \mu & -(\mu+3\lambda) & 3\lambda & 0 & 0 \\ 0 & 2\mu & -(2\mu+2\lambda) & 2\lambda & 0 \\ 0 & 0 & 3\mu & -(3\mu+\lambda) & \lambda \\ 0 & 0 & 0 & 4\mu & -4\mu \end{bmatrix} \tag{2.29}$$

Based on Equation (2.27) introduced in Section 2.3.2.3, we can get the steady-state probabilities of our given multiprocessor system as follows:

$$\boldsymbol{\pi} = \{\pi_0, \pi_1, \pi_2, \pi_3, \pi_4\} = \{\frac{\mu^4}{D}, \frac{4\lambda\mu^3}{D}, \frac{6\lambda^2\mu^2}{D}, \frac{4\lambda^3\mu}{D}, \frac{\lambda^4}{D}\}$$

where $D = \mu^4 + 4\lambda\mu^3 + 6\lambda^2\mu^2 + 4\lambda^3\mu + \lambda^4$.

Suppose the failure rate λ is $0.002h^{-1}$ and the repair rate μ is equal to $0.01h^{-1}$. Initially, the system is in state S_0. We use the uniformization method as described in Section 2.3.2.2 to compute the transition probability matrix $P(t)$ at times of 50 hours and 100 hours respectively, where ϵ is set to 0.00001:

$$P(50) = \begin{bmatrix} 0.7315 & 0.2379 & 0.0290 & 0.0016 & 0.0000 \\ 0.2974 & 0.5661 & 0.1263 & 0.0099 & 0.0003 \\ 0.1209 & 0.4210 & 0.3991 & 0.0568 & 0.0022 \\ 0.0492 & 0.2487 & 0.4261 & 0.2577 & 0.0183 \\ 0.0200 & 0.1327 & 0.3303 & 0.3654 & 0.1516 \end{bmatrix}$$

$$P(100) = \begin{bmatrix} 0.6094 & 0.3213 & 0.0635 & 0.0056 & 0.0002 \\ 0.4016 & 0.4469 & 0.1349 & 0.0159 & 0.0007 \\ 0.2647 & 0.4495 & 0.2409 & 0.0425 & 0.0024 \\ 0.1745 & 0.3984 & 0.3187 & 0.0999 & 0.0085 \\ 0.1150 & 0.3299 & 0.3549 & 0.1697 & 0.0304 \end{bmatrix}$$

It can be numerically verified that $P(100) = P(50)P(50)$. Besides, we compute the transient probabilities of the states $(S_0, S_1, \ldots, S_4)$ at the time of 100 hours:

$$\mathbf{p}(100) = \mathbf{p}(0)P(100) = \{0.6094,\ 0.3213,\ 0.0635,\ 0.0056,\ 0.0002\}$$

For example, the probability that this multiprocessor system is in state S_1 after *100* hours operation is 0.3212 based on the computed transient probabilities.

2.4 Related work

2.4.1 Performance analysis of NoCs

In the NoC literature, researchers have investigated the performance in both simulator-based and analytical ways. With help of simulation tools, NoC designers and researchers can explore the architectural design space in terms of some metrics such as performance, cost, and power. Because of the complexity of modern NoCs, it usually takes much time to implement a suitable simulator to simulate important aspects of such communication systems. Thus, analytical models are proposed as an alternative approach for performance analysis of NoCs.

2.4.1.1 Simulator based performance analysis of NoCs

In [109], the performances of different NoC topologies on a wormhole-switched simulator were compared and analyzed. Authors in [18] analyzed and compared the performance of NoCs of different topologies such as ring, spidergon, and mesh using an OMNET++ based simulator. They obtained throughput and latency of the evaluated topologies under simulation parameters of data injection rates, source and destination distributions, and a variable number of nodes. They concluded that spidergon topology is a good trade-off between performance and scalability. In [136], the authors utilized the NS-2 simulator to evaluate a NoC mesh topology that is based on the chip-level integration of communication heterogeneous elements. In [67], another research work about the mesh NoC has been conducted using the NS-2 simulator under the performance metrics such as latency and packet loss rate. They analyzed

how the communication load and the buffer size affected the performance of mesh NoC. Authors in [154] proposed an application-specific NoC design methodology to fit the requirements of some specific applications. They utilized the OPNET simulator to compare the proposed structure with a mesh topology under the application of a decoder SoC. In [50], authors performed a systematic performance analysis of 3D NoCs concerning latency, throughput, and wiring area overhead compared to traditional 2D implementations employing their cycle-accurate simulator. Authors in [54] used an open-source simulator named Booksim2 to implement a mesh-bus hybrid architecture for 3D NoCs. Moreover, performances of such 3D NoCs were evaluated concerning saturation throughput and zero-load latency.

2.4.1.2 Performance analysis of NoCs in an analytical way

The commonly used analytical performance evaluation methods are queuing theory, network calculus, and dataflow analysis.

Queuing theory

Queuing theory uses queuing models to represent different types of queuing systems. The commonly used notation to describe a queuing node is Kendall's notation in the form of *A/S/c/K/N/D*, where *A* denotes the distribution of the inter-arrivals, *S* is the service time distribution, *c* represents the number of service channels at the evaluated node, *K* denotes the maximum capacity of the queue, *N* is the size of the population of jobs to be served, and *D* is the queuing discipline [39].

Queuing theory has already been used in performance analysis of computer networks before it is applied for the performance analysis of NoCs. E.g. authors in [74] presented a queuing model for wormhole routing with finite-size buffers. To guarantee that circular link dependency would not occur, a deadlock-free routing scheme was assumed. Their proposed model estimated the network throughput in an interconnection network.

Researchers use queuing theory mainly to estimate the average performance of NoCs, such as average throughput, average packet latency, and average energy consumption. In general, system designers use these performance metrics to make decisions related to buffer or link capacity and application mapping. E.g. authors in [3] developed a close queuing network

for mesh-based interconnection networks with deterministic routing. They utilized approximate mean value analysis to compute average packet latency under arbitrary source-destination probability. A novel analytical model based on the M/M/1/K queuing model was proposed by Hu et al. [73] to quickly analyze and detect potential performance bottlenecks in different router channels. Besides, they developed a system-level buffer planning algorithm based on their analytical model to customize the router design. More specifically, their developed algorithm can automatically determine the buffer depth for each input channel in different routers on the NoC given the traffic characteristics of the target application and the total budget of the available buffering space. Based on the M/G/1 queuing model, authors in [105] presented how to compute the average number of packets at each buffer for wormhole-switched NoCs.

Apart from determining buffer capacity, the allocation of link capacities plays an important role in the design of NoC-based systems, as information traffic on such systems is heterogeneous. Authors in [63] presented a novel analytical delay model based on the M/M/1 queuing model for wormhole networks with virtual channels and non-uniform links. Moreover, they devised a capacity allocation algorithm based on their analytical analysis to allocate link capacities efficiently under specified performance requirements. In [106], the authors introduced a generic router model based on the probabilistic approach for NoC performance analysis under consideration of mapping application. The M/G/1/m queuing model was utilized in their work to find the blocking probability for a buffer at an input channel.

Queuing theory has also be used to study the performance of fault-tolerant routing algorithms and NoCs of different topologies. In [123], a new analytical model based on probabilistic analyses and queueing theory was developed for studying the performance of fault-tolerant routing algorithms in wormhole-switched 2D tori. Authors in [98] utilized queueing theory to study the traffic behavior of the spidergon NoC. In that work, message latency under different traffic rates was modeled and verified through simulations. In [126], authors developed an analytical model based on queueing theory to compute the latency of their proposed adaptive routing algorithm under a different number of elevators and different traffic patterns. The model presented by the authors in [27] was able to estimate the communication performance of arbitrary topology wormhole-switched NoCs with virtual channels. First, they utilized the routing path decomposition approach to obtaining a set of ordered link categories. Then, traditional queuing models such as M/M/1 and M/M/1/K were used to derive the transmission latency for each network component.

Network calculus

Le Boudec and Thiran [85] introduced the network calculus method to derive some useful properties (such as the worst-case bounds on maximum latency, backlog, and minimum throughput) in network-based systems. Moreover, network calculus can be seen as a theory for analyzing performance guarantees in the network. Traffic patterns and dynamic features of the network can be characterized with help of network calculus. Thus, it has also been widely used in the performance analysis of NoCs. Qian et al. [113] first presented an analysis procedure based on network calculus theory to calculate the delay bound for an individual flow interfered by other flows in a NoC. The authors first computed the equivalent service curve for individual flows with help of network calculus and then derived their packet delay bounds. As wormhole-switched NoCs are widely used in NoC research, these authors later extended their analysis procedure to be able to analyze communication delay bounds for individual flows in wormhole NoCs, in which not only link sharing but also buffer sharing needed to be considered [114]. In [88], network calculus was utilized to compute the delay and buffer bounds for time-division-multiplexing virtual circuits that crossed multiple clock domains. Authors in [13] proposed a network calculus-based methodology to analyze and evaluate the performance of NoCs of various topologies such as mesh, spidergon, and WK-recursive mesh. The used performance metrics were the end-to-end delay and buffer size requirements. They found that the WK-recursive mesh outperformed other topologies with respect to the end-to-end delay and buffer size requirements. In [115], authors demonstrated their network-calculus-based worst-case communication performance analysis for 3D NoCs. Besides, their results were verified by the simulations. Based on the experimental results, they found that 3D NoCs do not achieve better worst-case communication performance than 2D NoCs.

Dataflow analysis

A dataflow graph is a directed graph, in which nodes denote communication and arcs represent ordered sequences of tokens. It is generally used to describe a model-of-computation in which some concurrent processes utilize unbounded first-in-first-out (FIFO) channels to communicate with each other [75]. Various dataflow graphs have been proposed in the literature: synchronous data flow graph, cyclo-static dataflow graph, and dynamic dataflow

graph. Initially, dataflow graphs have been widely used in embedded systems. E.g. authors in [132] utilized synchronous dataflow graphs to model time-constrained multimedia applications. They proposed a novel resource allocation strategy that can bind multiple synchronous dataflow graphs to an embedded multiprocessor system under the given throughput guarantees to each application. Bekooij et al. [15] presented how dataflow models were used to derive the worst-case end-to-end temporal behavior of hard real-time applications and test statistical assertions about the temporal behavior of soft real-time applications. For NoCs, the authors in [66] used a cyclo-static dataflow graph to compute buffer sizes in the network interface for NoC applications, whose performance needs to be guaranteed.

In summary, the performance analysis of NoCs has been widely conducted in both simulation-based approaches and analytical approaches. However, the reliability of NoCs is not considered jointly in these works.

2.4.2 Reliability analysis of NoCs

Comparing to the performance analysis of NoCs, the reliability analysis of NoCs has not received much attention. However, reliability analysis has been widely studied for interconnection networks used in parallel systems. E.g. authors in [81] presented a decomposition-based analytical model for the reliability evaluation of hypercube multicomputer systems. A hypercube is an n-dimensional multicomputer network, in which each node represents a computer and the communication between the nodes is based on message passing protocol. Besides, communication topologies such as mesh, ring, or tree can be embedded in a hypercube. In their proposed approach, a hypercube of a high dimension was recursively decomposed into smaller hypercubes. First reliabilities of smaller cubes were computed, then the reliability of the evaluated hypercube was obtained based on the computation of the reliability of its decomposed cubes. Mahgoub [91] presented how to compute the reliability of the cluster-based systems through analytical modeling techniques. A cluster-based system contains processors, crossbar switches, buses, and memory modules. Based on their experimental results, they proposed better architecture for the cluster-based system to achieve higher reliability. Wang et al. [145] evaluated the reliability and availability of large mesh-based multicomputer systems under a more realistic model, in which each computer has

an independent failure probability. They proposed a novel technique to obtain lower bounds on the connectivity probability for mesh-based networks.

In the NoC domain, the work in [119] evaluated reliability in application-specific mesh-based NoC architectures. They define the path reliability for a source-destination pair as the product of the reliabilities of all routers along the path specified by XY routing. Authors in [99] proposed a model for computing the probability that a link of a NoC fails due to manufacturing variation. The impacts of link faults with respect to the cycles needed for communication were evaluated. In [31], an analytical model was proposed to assess the reliability factor of mesh NoC based SoC design with consideration of transient faults. The reliability factor is defined as the probability that faults can be recovered without any effect on the functionality of an evaluated system. Authors in [143] utilized some probabilistic and analytical models to evaluate the system-level reliability of mesh and torus NoCs with different parameters such as routing algorithms, traffic models, and network sizes. In the work [35], authors proposed a reliability assessment methodology for fault-tolerant NoCs. The methodology first divided the system into small subsystems. The reliability of each subsystem could be obtained using a state model. Finally, the reliability of the evaluated system could be computed from the obtained reliabilities of its subsystems. Formal reliability analysis for a resilient routing algorithm on 3D partially-connected NoCs was presented in [125]. However, these works only deal with reliability analysis, while the performance of NoCs under network faults are not taken into account.

2.4.3 Research of performability analysis

Performability analysis of computing and communication systems

Since the term performability was introduced by Meyer [94], researchers from different fields have studied and applied the performability. In this work, we discuss some research works related to computing and communication systems. Ghosh et al. [56] proposed an analytic model based approach to analyze the end-to-end performance of an infrastructure-as-a-service cloud under consideration of failures and repairs. Authors in [82] utilized a framework named CloudSim to evaluate the performability of an infrastructure-as-a-service cloud with help of introducing various failure modes and recovery

mechanisms. Ma et al. [90] presented two model-based techniques namely the exact composite model technique and the approximate model technique for analyzing the performability of a wireless communication network system. Based on numerical results, they found that an approximate model provided a remarkably accurate prediction of the evaluated system performance. In [6], the effect of access point failures on the performance of a WiFi communication system was studied through their proposed performability model, which used a continuous-time Markov chain to compute the steady-state probabilities. Then a penalty as a reward was computed for each state.

Performability analysis of NoCs

In the NoC literature, performability research can be classified into two categories. Works from the first category focus on various coding algorithms and error control schemes. For example, authors in [46] analyzed three error control schemes with joint consideration of performance, reliability, and energy consumption. For a specified time interval, the probability to successfully transmit a certain amount of bits in the presence of noise through a link of a NoC was defined as the interconnect-performability. Based on the definition of interconnect-performability from [46], traffic distribution matrix, and connectivity matrix, authors in [48] proposed the performability of network topology. The traffic distribution matrix specified the number of packets that need to be transmitted for each end-to-end communication flow. The connectivity matrix provided the number of links that need to be traversed by a packet from a source node to a destination node. Performabilities of several topologies were analyzed utilizing the proposed methodology. Authors in [127] defined performability as the probability of transmitting a correct flit. Different coding algorithms were analyzed with respect to the tradeoff between performability and energy. However, these works dealt with only the physical layer. How to model NoCs as a CTMC and use the Markov reward model for performability analysis are our concern and contribution. To our knowledge, this is the first work that analyzes NoCs based on the classical definition of performability.

Chapter 3
Markov modeling of NoCs

As we have already explained, a Markov reward model contains two parts: a continuous-time Markov chain and rewards. A stochastic process that satisfies the Markov property is a Markov chain [53]. In simple terms, a Markov chain is a stochastic process, whose future is only dependent on its current state instead of the whole past states. It allows us to model the uncertainty in many real-word systems that evolve dynamically in time [64].

The main focus of this chapter is about how to model NoCs as a CTMC. Generally, some assumptions are made to model a system. Based on our listed assumptions of NoCs in Section 3.1, a generic Markov state of 2D and 3D NoCs are presented. Then, we apply the proposed Markov state to three different use cases. In the first and second use cases, we focus on the 2D mesh-based NoCs. The first use case assumes that only routers can suffer from transient faults. Comparing to the first use case, we assume permanent faults can occur in both routers and links in the second one. Moreover, we presented a generic Markov model of mesh-based NoCs for each use case. The third use case takes into account the architecture of 3D vertically-partially-connected NoCs. Here, we focus on the permanent faults of links and TSVs. Finally, we use a 3D partially-connected mesh of size 4x4x3 to demonstrate our proposed Markov model.

As the size of NoCs increases, the size of their Markov state spaces grows large as well. To evaluate large size NoCs, we develop tools to generate our proposed Markov models and perform Markov model analysis conveniently.

3.1 CTMC model of NoCs

In our work, we focus on the architectures of 2D and vertically-partially-connected 3D NoCs. A vertically-partially-connected 3D NoC consists of several horizontal layers. Each horizontal layer can be viewed as a 2D NoC. Horizontal layers are connected utilizing the TSV technology. Faults can occur at links, routers, and TSVs [117]. To model NoCs as a CTMC, following assumptions are made:

- Routers may suffer from both transient and permanent faults.
- Links and TSVs suffer from only permanent faults.
- Both failure and repair time are exponential random variables.
- Links from different horizontal layers may have different failure rates. On the other hand, they have the same failure rate on the same horizontal layer.
- TSVs from different interlayers may have different failure rates. However, TSVs from the same interlayer have the same failure rate.
- Similarly, routers from different horizontal layers may have different failure rates. Moreover, on the same horizontal layer, routers may also have different failure rates.
- The maximum allowed number of faulty routers, links, and TSVs are limited. Each horizontal layer has the same router fault limit and the same link fault limit. The TSV fault limit of each interlayer is equal.

In this part, we present a generic state definition that can be applied for both 2D and 3D NoCs first. From the perspective of modeling, 2D NoC is a special case of 3D NoC, in which only one horizontal layer exists. A state in a CTMC represents a measurable characteristic of interest of a system at some time after it started the operation. In our case, a state contains information about the number of functioning routers, links, and TSVs of an evaluated NoC. The size of a 3D NoC is (x, y, z), in which x and y represent sizes of the dimension of a horizontal layer and z denotes the size of the vertical dimension. In case that z is set to 1, the evaluated 3D NoC becomes a 2D NoC. Furthermore, unlike the multiprocessor system illustrated in Section 2.3.2.4, routers from different positions on a horizontal layer may affect a NoC's performance differently. When we model a NoC as a CTMC, we want to take into account the positions of routers on each horizontal layer. Our proposed way to consider the positions of routers is classifying routers on a horizontal layer into different router groups.

As each horizontal layer has the same topology, it is assumed routers on each horizontal layer can be classified into K router groups based on some criterion. Together with the above-listed assumptions, we define a state s for the vertically-partially-connected 3D NoCs as follows:

$$
\begin{aligned}
s = (&r_{0,0}, r_{0,1}, \ldots, r_{0,j}, \ldots, r_{0,K-1},\\
&r_{1,0}, r_{1,1}, \ldots, r_{1,j}, \ldots, r_{1,K-1},\\
&\ldots,\\
&r_{i,0}, r_{i,1}, \ldots, r_{i,j}, \ldots, r_{i,K-1},\\
&\ldots,\\
&r_{z-1,0}, r_{z-1,1}, \ldots, r_{z-1,K-1},\\
&l_0, l_1, \ldots, l_i, \ldots, l_{z-1},\\
&t_0, t_1, \ldots, t_i, \ldots, t_{z-2})
\end{aligned}
\tag{3.1}
$$

where $r_{i,j}$ denotes the number of functioning routers from the j-th router group of the i-th horizontal layer, l_i represents the functioning links of the i-th horizontal layer, and t_i denotes the functional TSVs of the i-th interlayer. From implementation perspective, a state can be viewed as an integer array with length of $(z(K+2)-1)$. The index of $r_{i,j}$ is $(iK+j)$. Similarly, the indexes of l_i and t_i are $(zK+i)$ and $((z+1)K+i)$ respectively. Furthermore, all symbols utilized in the description of Markov modeling are summarized in Table 3.1.

A state is called a valid state represented by s_v if the following conditions are fulfilled:

- The number of faulty routers in each horizontal layer is not greater than the specified router fault limit: $\forall i \in \{0, 1, \ldots, z-1\}\ \Phi_r(i) <= n_r$
- And, the number of faulty links in each horizontal layer is not greater than the specified link fault limit: $\forall i \in \{0, 1, \ldots, z-1\}\ \Phi_l(i) <= n_l$
- And, the number of faulty TSVs in each interlayer does not exceed the specified TSV fault limit: $\forall i \in \{0, 1, \ldots, z-2\}\ \Phi_t(i) <= n_t$

Similarly, a state is called a failure state denoted by s_f if the following conditions are satisfied:

- The number of faulty routers in any horizontal layer is greater than the specified router fault limit: $\exists i \in \{0, 1, \ldots, z-1\}\ \Phi_r(i) > n_r$
- Or, the number of faulty links in any horizontal layer is greater than the specified link fault limit: $\exists i \in \{0, 1, \ldots, z-1\}\ \Phi_l(i) > n_l$
- Or, the number of faulty TSVs in any interlayer exceeds the specified TSV fault limit: $\exists i \in \{0, 1, \ldots, z-2\}\ \Phi_t(i) > n_t$

When any router, link, or TSV suffers from faults, the assessed NoC will move from one state to another subsequent one. Let s_{eval} represent the state that we want to evaluate. Besides, it is assumed to be a valid state. If a

Table 3.1: List of symbols for Markov modeling

Symbol	Description
$r_{i,j}$	the number of functioning routers from the j-th router group of the i-th horizontal layer
l_i	the number of functioning links on the i-th horizontal layer
t_i	the number of functioning TSVs on the i-th interlayer
K	the number of router groups
n_r	router fault limit
n_l	link fault limit
n_t	TSV fault limit
$\Phi_r(i)$	the total number of faulty routers on the i-th horizontal layer
$\Phi_l(i)$	the total number of faulty links on the i-th horizontal layer
$\Phi_t(i)$	the total number of faulty TSVs on the i-th interlayer
$\lambda_{r_{i,j}}$	the failure rate of routers from the j-th router group of the i-th horizontal layer
$\mu_{r_{i,j}}$	the repair rate of routers from the j-th router group of the i-th horizontal layer, if routers suffer from transient faults
λ_{l_i}	the failure rate of links from the i-th horizontal layer
λ_{t_i}	the failure rate of TSVs from the i-th interlayer
$N_{i,j}$	the maximum number of functioning routers from the j-th router group of the i-th horizontal layer
$tr(s_i, s_j)$	the transition rate from state s_i to state s_j

component from state s_{eval} becomes faulty and the index of its corresponding element in s_{eval} is P, the subsequent state represented by s_{sub} of the state s_{eval} is defined as follows:

$$s_{sub}(x) = \begin{cases} s_{eval}(x), & \text{if x} \neq \text{P} \\ s_{eval}(x) - 1 & \text{if x} = \text{P} \end{cases} \tag{3.2}$$

where $x \in \{0, 1, ..., (z(K+2)-2)\}$.

Theoretically, the maximum number of subsequent states of a valid state is $(z(K+2)-1)$, which is also the length of a state. Among those subsequent states, zK of them are generated because of router faults, z of them are generated due to link faults, and the rest $(z-1)$ of them are introduced by TSV faults. Once the assessed NoC enters into a failure state, its performance will

not be taken into account. Therefore, a failure state does not have subsequent states. The transition rates between the state s_{eval} and its subsequent state s_{sub} can be computed as follows:

- When any router of the j-th router group on the i-th horizontal layer suffers from faults, s_{eval} will move to a subsequent state denoted by s_{sub} with a rate of $r_{i,j}\lambda_{r_{i,j}}$. If the state s_{sub} is still valid, routers suffer from transient faults, and the faulty routers can be repaired independently, it can be moved to the state s_{eval} with the rate of $(N_{i,j} - (r_{i,j} - 1))\mu_{r_{i,j}} = (N_{i,j} + 1 - r_{i,j})\mu_{r_{i,j}}$. However, if the state s_{sub} is not a valid state or the assumed fault model is permanent, it cannot move back to the state s_{eval}. Thus, the transient rate is 0.

$$tr(s_{sub}, s_{eval}) = \begin{cases} (N_{i,j} + 1 - r_{i,j})\mu_{r_{i,j}}, & \text{if } s_{sub} \text{ valid} \wedge \text{transient fault model} \\ 0 & \text{others} \end{cases} \tag{3.3}$$

- Similarly, when any link of the i-th horizontal layer suffers from faults, the state s_{eval} will move to a subsequent state s_{sub} with a rate of $l_i\lambda_{l_i}$. It is assumed that links suffer from permanent faults only, therefore, there is no transition from s_{eval} back to s_{eval}.
- Finally, when any TSV of the i-th interlayer suffers from faults, the state s_{eval} moves forward to a subsequent state s_{sub} with a rate of $t_i\lambda_{t_i}$. Besides, there is no transition from s_{eval} back to s_{eval} as TSVs suffer from only permanent faults.

In the following sections, we present how to apply the introduced state definition to different use cases under consideration of different fault models.

3.2 Use case: Markov modeling of 2D mesh-based NoCs under transient faults

Transient faults in the router are mainly caused by radiation effects. They can be represented by a pulse with the duration subject to some distribution [117]. E.g. authors in [156] assume that the duration of a pulse characterizing some transient faults is exponentially distributed. Some mechanisms such as time redundancy technique introduced in [8, 104, 47] can detect the transient faults. In our work, if a router suffers from some transient faults, it will be viewed as faulty and its status can be known to its neighbors.

In the first use case, we focus on the architecture of 2D mesh-based NoCs. We only consider the faults of routers. It means that faults in links do not need to be considered. Besides, we make the following assumptions:

- Routers are assumed to suffer only from transient faults.
- Each router has its local repair process. The local repair can be thought of as the disappearance of a transient fault. The disappearance requires some time.
- An evaluated NoC has a global repair process. The global repair can be thought of as a reset and restores a NoC from any failure state to the initial state.

Therefore, the generic state defined in Equation (3.1) is instantiated as follows:

$$s = (r_0, r_1, \ldots, r_{K-1}) \tag{3.4}$$

where r_i denotes the number of functioning routers of the i-th router group.

To find the state space of 2D mesh-based NoCs, the first step is to determine the router groups. Our proposed way is from the perspective of NoC topology, which can be described mathematically using a graph that contains a vertex set and an edge set. For the NoC, a vertex denotes a router. An edge between two routers contains two physical channels. One is for sending packets and the other one is for receiving packets. Vertex degree means the number of incident edges. Routers are classified into different groups based on their vertex degrees. The actual number of operational routers from the classified groups form a state. As mesh topology has three different vertex degrees, routers can be classified into three different groups: $Group_0$, $Group_1$, and $Group_2$. For example, Figure 3.1 illustrates how we classify routers on a mesh-based NoC of size 4x4 into three different router groups.

Therefore, K is equal to 3. The routers from $Group_0$ locate at the four corners. The maximum size of $Group_0$ is 4. Routers that sit in outer edges belong to $Group_1$. The other ones that locate inside the mesh are classified as $Group_2$.

After we have determined the state for mesh-based NoCs, the next step is to find out the transitions between possible states with consideration of failures, repairs and fault limit of routers. Our proposed Markov model of a mesh-based NoC is illustrated in Figure 3.2. The failure and repair rates related to routers from $Group_0$ are λ_0 and μ_0. Similarly, for routers from $Group_1$ and $Group_2$ they are λ_1, μ_1 and λ_2, μ_2 respectively. The $Phase\ x$ contains such states, whose total number of faulty routers is equal to x. When a certain number of routers are faulty, the NoC's performance may become so bad that it is not

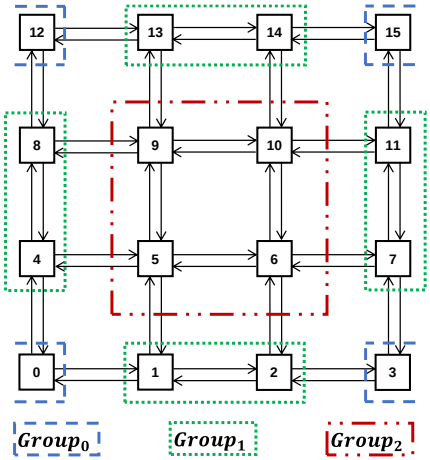


Fig. 3.1: Router groups of a mesh-based NoC of size 4x4

suitable for fulfilling communication tasks. Thus, the total allowed number of faulty routers needs to be limited. It is denoted by n_r. A state, in which the total number of faulty routers exceeds n_r, is called a failure state. In this figure, we assume that the router fault limit n_r is at least 4.

We briefly describe the state transitions in the model shown in Figure 3.2 as follows:

- When a router from a group suffers from faults, a state moves forward to a subsequent state. E.g. for the situation that a fault in $Group_2$ happens, it makes the model move from state (r_0, r_1, r_2) to state $(r_0, r_1, r_2 - 1)$ with a rate of $r_2\lambda_2$.
- When a repair for a faulty router from a group is complete, a state moves backward to a previous state in case that the state is not a failure state. E.g. in state $(2, r_1, r_2)$ two routers from $Group_0$ are faulty. As each router has its own independent repair process, this state goes back state $(3, r_1, r_2)$ with a transition rate of $2\mu_0$. If the state is a failure state, the global repair will restore the NoC to its initial state.

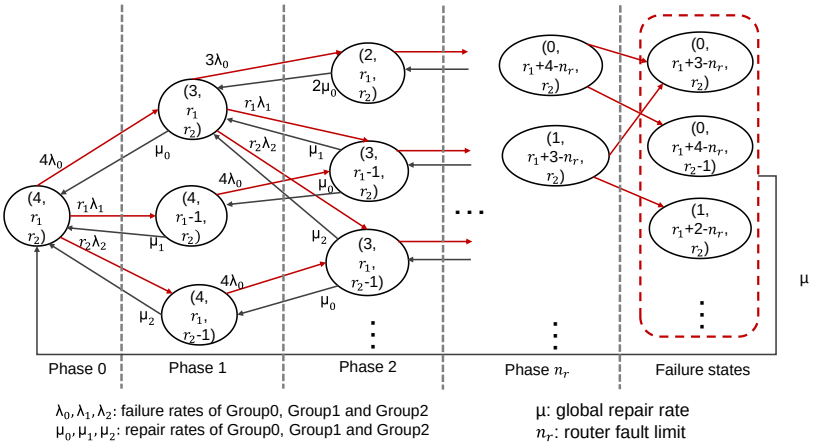


Fig. 3.2: Markov model of a mesh-based NoC under consideration of transient faults

3.3 Use case: Markov modeling of 2D mesh-based NoCs under permanent faults

A permanent fault will not disappear once it occurs. As an example, a broken link on a NoC can be viewed as a permanent fault. Permanent faults are mainly caused by structural defects. Structural defects may occur during the chip processing process [147]. Besides, they can also happen during operation because of aging. The aging problem is caused by several physical effects such as electro-migration, bias temperature instability, hot carrier injection, and time-dependent dielectric breakdown [117]. Permanent faults can be located and detected. E.g. built-in-self test is broadly used to detect permanent faults [153]. In this thesis, we suppose that the permanent fault of a component can be detected and known to its connected neighbor routers.

We still focus on the 2D mesh-based NoCs in the second use case. Comparing to the first one, we make the following assumptions:

- Routers and links can only suffer from permanent faults, which will appear while a NoC's operation goes on. The lifetime of a router or a link follows an exponential distribution.

- If the number of faulty routers or the number of faulty links exceeds the specified corresponding limit, the system moves into a failure state.
- Classify routers into 3 router groups based on their vertex degrees as described in Section 3.2.

The generic state is instantiated for the second use case in such a way:

$$s = (r_0, r_1, r_2, l) \tag{3.5}$$

where r_0, r_1, and r_2 denote the numbers of operational routers from router groups $Group_0$, $Group_1$, and $Group_2$. Similarly, l represents the number of functioning links.

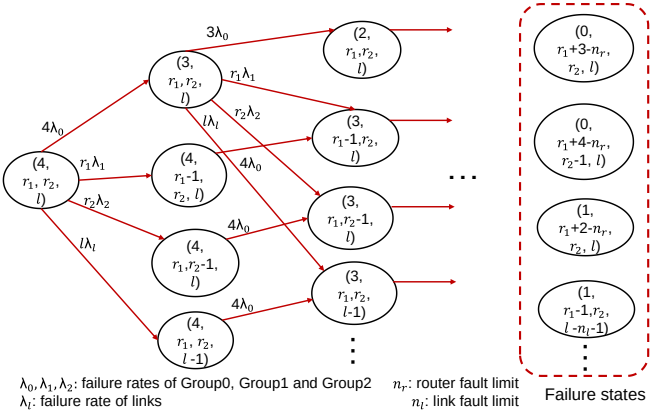


Fig. 3.3: Markov model of a mesh-based NoC under consideration of permanent faults

Our proposed model for a 2D mesh-based NoC under consideration of permanent faults both in routers and in links is illustrated in Figure 3.3. Here, we assume that the router fault limit n_r is at least 4. λ_0, λ_1, and λ_2 represent failure rates of routers from router groups: $Group_0$, $Group_1$, and $Group_2$ respectively. Besides, λ_l is the failure rate of links. When a fault occurs in a router or a link, a state transition happens. E.g. in case that a router from $Group_0$ suffers from a fault in the state $(3, r_1, r_2, l)$, it makes the system move from this state to the state $(2, r_1, r_2, l)$ with a rate of $3\lambda_0$. States from our

proposed model are classified into two groups: valid states and failure states. Once the system moves into a failure state, its performance will be treated as meaningless. Besides, a failure state does not have any subsequent states in comparison to a valid state.

3.4 Use case: Markov modeling of 3D partially-connected mesh-based NoCs

In the third use case, we assume that links in horizontal layers and TSVs in interlayers suffer from permanent faults, which will appear while a NoC's operation goes on. Moreover, their failure rates under exponential distribution are constant. A link consists of two channels. One channel is for sending packets, and the other one is for receiving packets. In case that a link is faulty, a packet cannot traverse through it anymore. Similarly, the faulty TSVs are discussed in a form of bidirectional faulty links through which a packet cannot traverse in the upward and downward directions. In addition, it is assumed that links in each layer and TSVs in each interlayer may have different failure rates. Therefore, a Markov state can be viewed as a tuple that stores the current number of functional links in each layer and TSVs in each interlayer. If the size of a 3D mesh is (x, y, z), in which x and y represent sizes of dimensions of a horizontal layer and z denotes size of the vertical dimension, the length of a state is $(2z - 1)$. The generic state is instantiated for the third use case as follows:

$$s = (l_0, l_1, \ldots, l_{z-1}, t_0, t_1, \ldots, t_{z-2}) \tag{3.6}$$

where l_i represents the functional links of the i-th horizontal layer and t_i denotes the functional TSVs of the i-th interlayer.

For the third use case, we use a 3D partially-connected mesh of size 4x4x3 as illustrated in Figure 3.4 to demonstrate how to model it as a CTMC. Three horizontal layers (layer0, layer1, and layer2) and two interlayers (interlayer0 and interlayer1) exist in this illustrated 3D mesh. There are 24 links in each horizontal layer and 4 TSVs per interlayer. Figure 3.5 illustrates our proposed Markov model. The failure rates related to links from layer0, layer1 and layer2 are represented by λ_{l_0}, λ_{l_1} and λ_{l_2} respectively. Furthermore, λ_{t_0} and λ_{t_1} denote the failure rates of TSVs from interlayer0 and interlayer1. When a certain number of links or TSVs are faulty, a 3D partially connected mesh-based NoC may become so congested that it is not suitable for fulfilling communication

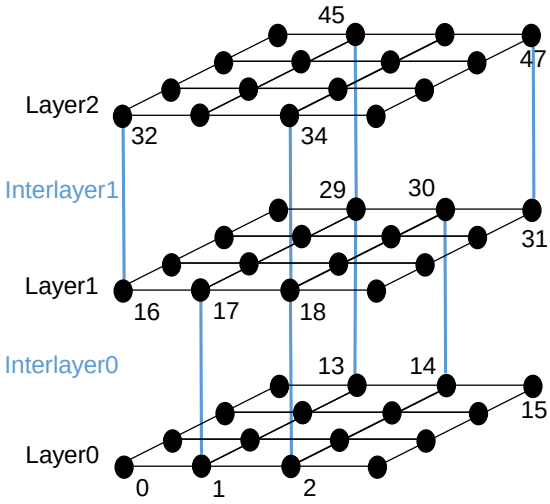


Fig. 3.4: An example of a vertically-partially-connected 3D mesh of size 4x4x3

tasks. Therefore, the allowed number of faulty links in each horizontal layer or TSVs in each interlayer needs to be limited. In our work, it is assumed that the maximum allowed number of faulty links n_l is 25 percent of total links in a horizontal layer. Similarly, the maximum allowed number of faulty TSVs n_t is 50 percent of total TSVs in an interlayer. Moreover, it is assumed that the positions of functional TSVs can be detected and are known to all routers in this network.

When a link or a TSV suffers a fault, a state transition happens. E.g. in case that a fault in the first layer happens in the initial state, it makes the model move from the state $(24, 24, 24, 4, 4)$ to the state $(23, 24, 24, 4, 4)$ with rate of $24\lambda_{l_0}$. States from our proposed Markov model are divided into two groups: valid states and failure states. A state, in which the number of faulty links in each horizontal layer or the number of faulty TSVs in each interlayer does not exceed the corresponding maximum allowed limit, is called a valid state. Otherwise, it is called a failure state. Once a failure state is entered, performance of the evaluated network will be treated as meaningless. E.g. the

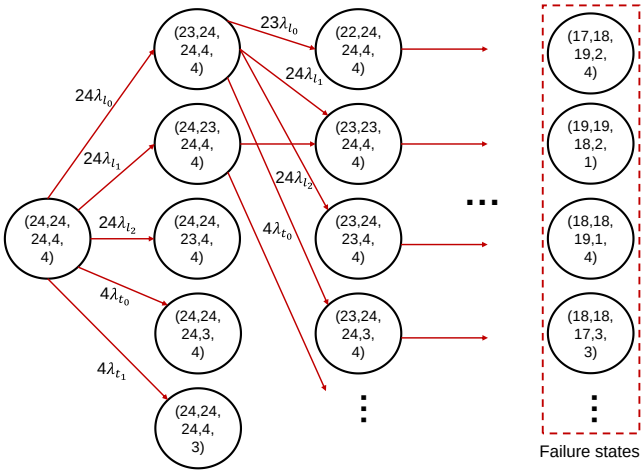


Fig. 3.5: Markov model of a partially-connected 3D mesh of size 4x4x3

state $(17, 18, 19, 2, 4)$ is a failure state, as the number of faulty links in its first layer exceeds the allowed maximum number of faulty links. Furthermore, a failure state does not have any subsequent states.

3.5 Tools for Markov model generation and analysis

3.5.1 Markov model generator

Until now, we have presented how to model NoCs as a CTMC. As the size of NoCs increases, the state space of their Markov models becomes large as well. To evaluate the large size NoCs, we developed a tool to generate the Markov states and the generator matrix for the evaluated NoCs. Figure 3.6 depicts the structure of our developed tool named Markov model generator. Its main functionalities are listed as follows:

1. Generate valid and failure states
2. Create the generator matrix for the evaluated NoC
3. Save the generated states and generator matrix as text files

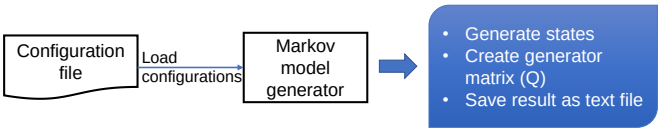


Fig. 3.6: Structure of Markov model generator

The configuration parameters are stored in a configuration file in the format of YAML [1] and listed as follows:

- Size of router groups (router groups of different horizontal layers have the same size)
- Initial numbers of routers from specified router groups of different horizontal layers
- Failure rates of router groups of different horizontal layers
- Repair rates of router groups of different horizontal layers (In case that permanent faults are considered, we set all repair rates to zero)
- Global repair rate (for use cases, in which only transient router faulty are considered)
- Initial numbers of links of different horizontal layers
- Failure rates of links of different horizontal layers
- Initial numbers of TSVs of different interlayers
- Failure rates of TSVs of different interlayers
- Fault limits of routers, links, and TSVs

The detailed implementation of the procedure of creating the generator matrix is depicted in Algorithm 1, where:

- C: configuration parameters
- s_0: the initial state
- S_v: set of valid states
- S_f: set of failure states
- Π: the dictionary storing states and their corresponding state numbers. State is the key. E.g. the state number of s_0 is zero
- Q: the generator matrix that is a two-dimensional vector
- K: the subsequent states of a valid state
- μ: the global repair rate

It first creates the generator matrix based on the valid and failure states and initializes all its elements to zeros. For each valid state, we compute its subsequent states. The transition rate to each subsequent state is calculated and stored in its corresponding position in the generator matrix. The position is found with the help of the state dictionary Π. For such a subsequent state, which is also a valid one, a repair rate will be computed and stored in the generator matrix. With this repair rate, the valid subsequent state k can move back to the valid state s. The procedures of generating subsequent states and computing the transition rates between states have been described in Section 3.1. In case that the global repairs are taken into account, it means that a failure state can move back to the initial state with a rate of μ.

Algorithm 1 Procedure of creating generator matrix

```
1:  procedure CreateGeneratorMatrix()
2:      C ← loadConfiguration(configFile)
3:      S_v ← generateValidStates(C)
4:      S_f ← generateFailureStates(C)
5:      Q ← initializeQ(S_v, S_f)
6:      Π ← createStateDictionary(S_v, S_f)
7:      for each s ∈ S_v do
8:          K ← generateSubsequentStates(s)
9:          for each k ∈ K do
10:             Q[Π[s]][Π[k]] ← computeFailureRate(C, s, k)
11:             if isValidState(k) then
12:                 Q[Π[k]][Π[s]] ← computeRepairRate(C, s, k)
13:             end if
14:         end for
15:     end for
16:     if μ > 0 then
17:         for each s ∈ S_f do
18:             Q[Π[s]][Π[s_0]] ← μ
19:         end for
20:     end if
21:     computeDiagonalEntry(Q)
22: end procedure
```

For example, we applied our Markov model generation tool to generate valid and failure states as depicted in Section 3.2 for different sizes meshes. We set the fault limit as 10 percent of the total number of routers. Table 3.2 provides the information of states for different size meshes. As the mesh's size increases, its Markov state space grows but remains tractable.

Table 3.2: Fault limit (10 percent) and states information for meshes with different sizes under consideration of transient faults of routers

Mesh	Fault limit	No. of total states	No. of valid states
36	4	55	35
64	7	145	110
100	10	280	230
144	15	605	530
196	20	1055	955

3.5.2 Markov model analysis

Fig. 3.7: Structure of Markov model analysis tool

To deal with the large size of an evaluated NoC's Markov state space, we implement a tool to analyze the utilized Markov model. The structure of our Markov analysis tool is depicted in Figure 3.7. It has the following main functionalities based on the provided generated matrix:

1. Compute the steady-state probabilities of the states of the Markov model of an evaluated NoC
2. Compute the transient probability matrix $P(t)$
3. Compute the transient probabilities of the states of the Markov model of an evaluated NoC

Our tool is implemented in MATLAB. The computation of the steady-state probabilities is based on Equation (2.27) and Equation (2.28). We implement the uniformization method as explained in Section 2.3.2.2 to compute the transient probability matrix $P(t)$. The algorithmic description of our implementation is depicted in Algorithm 2.

Algorithm 2 Procedure of transient probability matrix P(t)

procedure $ComputeTransientProbabilityMatrix(Q, t, \epsilon)$
  $r \leftarrow \max_{0 \le i \le M} \{|q_{i,i}|\}$
  $\hat{T} \leftarrow I + \frac{Q}{r}$
  $P = e^{-rt} I$
  $A = \hat{T}$
  $c \leftarrow e^{-rt}$
  $k \leftarrow 1$
  $sum \leftarrow c$
  while $sum < 1 - \epsilon$ **do**
    $c \leftarrow \frac{c*(rt)}{k}$
    $P \leftarrow P + cA$
    $A \leftarrow A\hat{T}$
    $sum \leftarrow sum + c$
    $k \leftarrow k + 1$
  end while
  return P
end procedure

Generally, ϵ is set to 0.00001. After the transition probability matrix $P(t)$ is computed, the transient probabilities of the states from the utilized Markov model at time t can be computed using Equation (2.22) given the initial distributions of the states.

In our work, we focus on two types of performabilities: long-term performability and transient performability. For long-term performability evaluation of a NoC, the long-term steady-state probabilities and the corresponding rewards of the valid states from its Markov model are required. Similarly, the transient probabilities and the rewards of the valid states from its Markov model are needed for transient performability evaluation. Up to now, we present how to model a NoC as a CTMC, compute steady-state or transient probabilities of states from the used Markov model of an evaluated NoC. We will present how to compute rewards for the utilized Markov model of a NoC in Chapter 4.

Chapter 4
Performance metrics and their computation

By now, we have already presented how to model NoCs as a CTMC. To perform performability analysis, we need two parts: a CTMC and performance metrics for its valid states. From Chapter 3, we know that a Markov model of a NoC consists of many states usually. Therefore, efficient computation of performance metrics is a new challenge for the performability analysis of large size NoCs. This chapter mainly focuses on performance metrics and how to compute them efficiently. In Section 4.1, a performance metric named communication time and an analytical approach to computing it is presented. Section 4.2 introduces another performance metric named fault resilience. Moreover, we propose an approach based on the channel dependency matrix to compute it faster. A machine learning based approach to estimating communication time and fault resilience is demonstrated by a mesh-based NoC of size 8x8 in Section 4.3. Finally, we present a multi-threading based tool to accelerate computations of performance metrics for valid states in Section 4.4.

4.1 Communication time

In this section, we introduce the definition of communication time first. Then, an analytical approach to compute communication time is proposed and verified. Communication time can be interpreted as the time to finish the specified communication load. As the purpose of a NoC is to perform communication between its connected computing cores or memories, communication time is a fair metric to assess the performance of a NoC. Furthermore, we demonstrate our proposed approach to computing communication time with mesh-based NoCs.

4.1.1 Definition of communication time

The definition of communication time is based on another two definitions: communication flow and communication round. Their relationships are illustrated in Figure 4.1. In other words, communication time is the time to fulfill a communication task, which requires many communication rounds normally. Besides, one communication round consists of some communication flows.

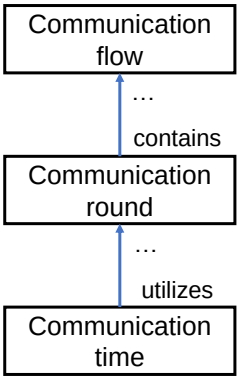


Fig. 4.1: Relationship between communication time, communication round, and communication flow

In our model, a node is defined as a processing core that is only connected to one router on a NoC. The router and its connected node share the same index. E.g. node 0 is connected to router 0. All symbols used in the description are summarized in Table 4.1.

4.1.1.1 End-to-end communication flow

The communication between any two different nodes connected to routers $i, j \in \Delta$ is defined as an end-to-end communication flow $f_{i \to j}$, where i denotes the source and j denotes the sink.

Table 4.1: List of symbols for communication time

Symbol	Description
Δ	Set of router ids
$f_{i \to j}$	An end-to-end communication flow from node i to node j
Π_i	The i^{th} communication round
$n(\Pi_i)$	The number of successfully delivered packets in the i^{th} communication round
t_{inj}	Injection delay from a source node to a source router
t_{ej}	Ejection delay from a destination router to a destination node
$\sigma(R_i \to R_j)$	Network latency from router i to router j
$\sigma(f_{i \to j})$	Latency of an end-to-end communication flow from node i to node j
$\sigma(\Pi_i)$	Latency of the i^{th} communication round
CT	Communication time
N	The specified number of packets for which communication time is determined
$C_{R_i \to R_j}$	Set of channels that make up a path from router i to router j
$\lvert C_{R_i \to R_j} \rvert$	Hop count from router i to router j
b	Physical channel bandwidth
b_S	Shared channel bandwidth
t_R	Routing computation delay of a head flit
t_S	Switching delay of a flit
t_{router}	Time to traverse a router. For a head flit, $t_{router} = t_R + t_S$. For body and tail flits, $t_{router} = t_S$
$t_{channel}$	Time for a flit to traverse a channel
c_B	Bottleneck channel that has minimum shared channel bandwidth

4.1.1.2 Communication round

A communication round $\Pi = \{f_{i \to j} : i, j \in \Delta\}$ is a set of end-to-end communication flows, in which all source nodes are different and every node delivers one packet to its destination that is selected based on the utilized traffic pattern. All packets have the same number of flits. Moreover, it is assumed that all source nodes send packets at the same time. A communication round in which the set of sources is a strict subset of Δ, is called a partial communication round. Similarly, if the set of sources in a communication round is equal to Δ, it is defined as a full communication round.

4.1.1.3 Latency of an end-to-end communication flow

Latency of an end-to-end communication flow is defined as the time span from the moment a packet is generated at a source node to the time when it is delivered to a destination node. It is the sum of injection delay, network latency and ejection delay. Formally, it is expressed as follows:

$$\sigma(f_{i\rightarrow j}) = t_{inj} + \sigma(R_i \rightarrow R_j) + t_{ej} \tag{4.1}$$

Injection delay is defined as the time, which is taken by a packet to pass through the channel from the source node to the source router. Network latency is the time that a packet takes to traverse from the source router to the destination router. Ejection delay is the time that a packet needs for crossing the channel between the destination router and the destination node. Additionally, the queuing delay at injection is zero because packets are injected into an empty network.

Figure 4.2 illustrates an end-to-end communication flow $f_{3\rightarrow 1}$ on a mesh-based NoC of size 3x3. Its latency is computed as follows:

$$\sigma(f_{3\rightarrow 1}) = t_{inj_3} + \sigma(R_3 \rightarrow R_1) + t_{ej_1}$$

4.1.1.4 Latency of a communication round

Latency of a communication round is the time that a NoC needs for delivering all packets in this round. It is determined by the maximum latency of all involved flows.

$$\sigma(\Pi_i) = \max_{\forall f \in \Pi_i} \sigma(f) \tag{4.2}$$

Figure 4.3 depicts a partial communication round on a mesh-based NoC of size 3x3, in which three end-to-end communication flows exist. These communication flows are: $f_{3\rightarrow 1}$, $f_{5\rightarrow 1}$, and $f_{6\rightarrow 2}$. The latencies of these communication flows can be calculated as follows:

$$\begin{aligned}
\sigma(f_{3\rightarrow 1}) &= t_{inj_3} + \sigma(R_3 \rightarrow R_1) + t_{ej_1} \\
\sigma(f_{5\rightarrow 1}) &= t_{inj_5} + \sigma(R_5 \rightarrow R_1) + t_{ej_1} \\
\sigma(f_{6\rightarrow 2}) &= t_{inj_6} + \sigma(R_6 \rightarrow R_2) + t_{ej_2}
\end{aligned}$$

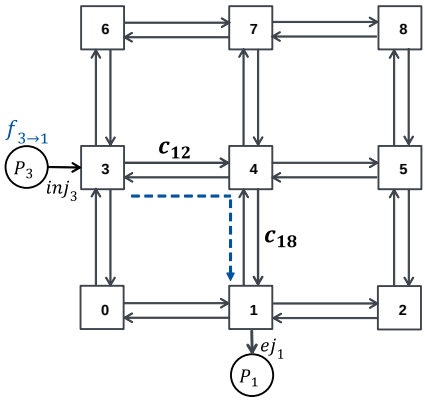


Fig. 4.2: Example of an end-to-end communication flow on a mesh-based NoC of size 3x3

The latency of this partial communication round is equal to the largest latency of $\sigma(f_{3\to1})$, $\sigma(f_{5\to1})$, and $\sigma(f_{6\to2})$.

4.1.1.5 Communication time

Communication time is the sum of the latencies of all utilized full communication rounds for successfully delivering the specified number of packets N. If the number of totally needed communication rounds is M, the communication time is defined as follows:

$$CT = \sum_{i=1}^{M} \sigma(\Pi_i),\ where \sum_{i=1}^{M-1} n(\Pi_i) < N \wedge \sum_{i=1}^{M} n(\Pi_i) \geq N \tag{4.3}$$

The usage of many full communication rounds enables us to assess a NoC's performance thoroughly, because different source-destination pairs are used in each round.

In order to calculate communication time, the problem of computing the latency of an end-to-end communication flow needs to be solved first. As different flow control methods may produce different network latencies, wormhole-

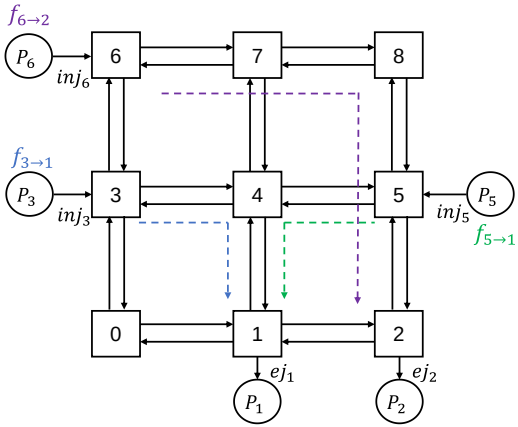


Fig. 4.3: Example of a communication round on a mesh-based NoC of size 3x3

switched flow control method is assumed in our model. In this method, each message is divided into packets, and each packet is further divided into a fixed number of flits. A packet encompasses a head flit, a tail flit, and some body flits. To facilitate constructing the model, the following assumptions are made:

1. A port of a router contains one input unit and one output unit. Each input unit consists of only one first-in-first-out (FIFO) buffer, which is applied to store incoming flits. All routers in the network need the same time to make routing computation and switching.
2. Physical channels have the same bandwidth b. If the time a flit traverses a channel is $t_{channel}$, t_{inj} and t_{ej} are equal to $t_{channel}$.

4.1.2 Zero-load latency

The network latency of a wormhole switched packet contains two parts: the latency of the head flit and the latency of the subsequent flits. The latency of the head flit is determined by router delay t_{router}, channel delay $t_{channel}$ and hop count $|C_{R_i \to R_j}|$ between the source and destination routers. A router

uses the head flit for routing computation. Then it transmits the head flit to the next router. Therefore router delay is composed of routing computation and switching for the head flit. Other subsequent flits follow the head flit through the NoC in a pipelined manner. Their latencies are determined by their size and maximum delay from switching and channel [41].

In general, the network delay of an m-flit packet is calculated as follows:

$$\begin{aligned}\sigma(R_i \rightarrow R_j) = [(|C_{R_i \rightarrow R_j}| + 1) \times (t_R + t_S) \\ +(|C_{R_i \rightarrow R_j}|) \times t_{channel}] \\ +[\max(t_S, t_{channel}) \times (m-1)]\end{aligned} \tag{4.4}$$

Equation (4.1) of latency of an end-to-end communication flow is rewritten as follows:

$$\begin{aligned}\sigma(f_{i \rightarrow j}) = 2 \times t_{channel} + \sigma(R_i \rightarrow R_j) \\ = [(|C_{R_i \rightarrow R_j}| + 1) \times (t_R + t_S) \ + (|C_{R_i \rightarrow R_j}| + 2) \times t_{channel}] \\ +[\max(t_S, t_{channel}) \times (m-1)]\end{aligned} \tag{4.5}$$

Figure 4.2 depicts an example of an end-to-end communication flow $f_{3 \rightarrow 1}$ on a mesh-based NoC. We assume that $t_R = t_S = 1\ cycle,\ b = 1\ flit/cycle$ and a packet contains 3 flits. A time-space diagram shown in Figure 4.4 confirms that its latency is 12 cycles. According to Equation (4.5), we compute the latency of communication flow $f_{3 \rightarrow 1}$ as follows:

$$\begin{aligned}\sigma(f_{3 \rightarrow 1}) = [(|C_{R_3 \rightarrow R_1}| + 1) \times (t_R + t_S) + (|C_{R_3 \rightarrow R_1}| + 2) \times t_{channel}] \\ +max(t_S, t_{channel}) \times (m-1) \\ = [(2+1) \times 2 + (2+2) \times 1] + 2 = 12\end{aligned}$$

where $|C_{R_3 \rightarrow R_1}| = 2$, $t_{channel} = 1$ cycle and $m = 3$. As can be seen, the computed result based on Equation (4.5) is the same as the result from the illustrated time-space diagram.

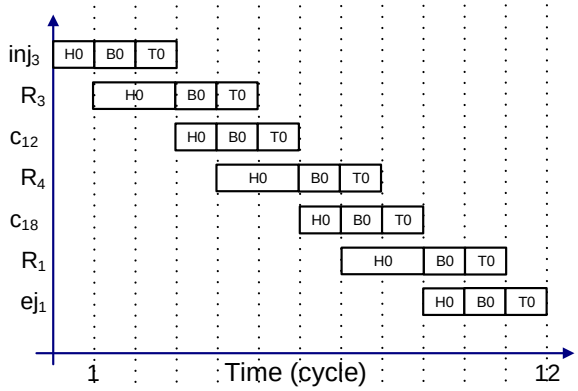


Fig. 4.4: Time-space diagram of an end-to-end communication flow on a mesh-based NoC of size 3x3

4.1.3 End-to-end communication flow latency with multiple flows in NoC

If more than one end-to-end communication flow sends packets at the same time, they may overlap at some channels, which leads to network contention.

4.1.3.1 Shared channel bandwidth

In order to consider the effect of network contention, we introduce a new concept named shared channel bandwidth. Figure 4.5 depicts a communication round consisting of three flows: $f_{3\to1}$, $f_{5\to1}$ and $f_{7\to1}$. We make two assumptions for this network: it uses XY routing and each channel's physical bandwidth is 1 $flit/cycle$. All these flows arrive at Router R_4 at the same time and want to use its south output channel c_{18}. Two different arbitrations can be used to solve their conflicts: winner-takes-all and interleaving. In both cases, the south output channel needs the same total time for transmitting the three flows. The effect is that this channel's bandwidth is shared by them. For each flow, the shared channel bandwidth of c_{18} is $\frac{1}{3}$ $flit/cycle$. Intuitively, if a channel with physical bandwidth b is used by n flows, its shared channel

bandwidth is computed as:

$$b_S = \frac{b}{n} \tag{4.6}$$

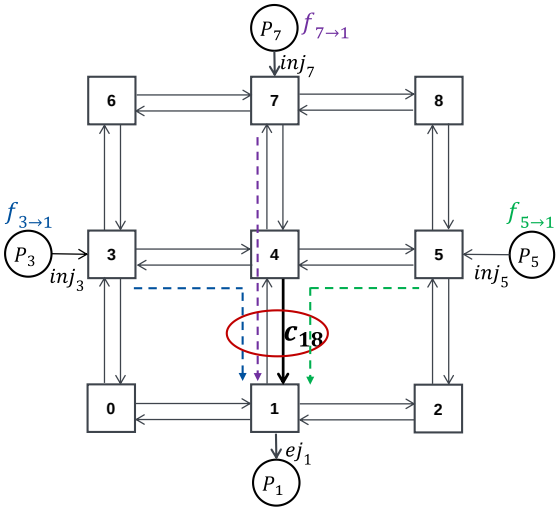


Fig. 4.5: Flows arrive at Router R_4 at the same time

4.1.3.2 Refined computation of shared channel bandwidth

Results from Equation (4.6) are pessimistic in some cases. Figure 4.6 illustrates another communication round containing three flows: $f_{3\to9}$, $f_{4\to13}$, and $f_{7\to9}$. XY routing is used on this network, and every channel has same physical bandwidth 1 $flit/cycle$. All these three flows use channel c_{23}, whose source router is R_5 and target router is R_9. Therefore, the shared channel bandwidth of c_{23} is $\frac{1}{3}$ $flit/cycle$. However, the hop counts from R_3, R_4, and R_7 to R_5 are 3, 1 and 2 respectively. The header flits of these flows arrive at R_5 at different time. In the beginning, $f_{4\to13}$ uses c_{23} without contention. When flits of $f_{7\to9}$ arrive at R_5, $f_{4\to13}$ and $f_{7\to9}$ have to compete for c_{23}. After flits of $f_{3\to9}$ arrive

at R_5, these three flows have contention at c_{23} for some time. In this case, $f_{7\to9}$ and $f_{3\to9}$ contribute only parts of their flits to contention on c_{23}. It means the shared channel bandwidth of c_{23} should be larger than $\frac{1}{3}\ flit/cycle$.

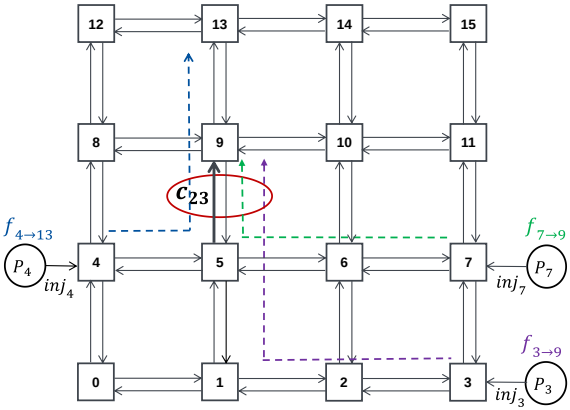


Fig. 4.6: Flows arrive at Router R_5 at different time

In order to refine calculation of a channel's shared bandwidth, the positions of flows' source routers that use the channel should be taken into account. Algorithmic description of our proposed refinement procedure is given in Algorithm 3, where:

- c – Channel, whose shared channel bandwidth will be refined
- $\Lambda(c)$ – Set of end-to-end communication flows in a communication round uses channel c
- $|\Lambda(c)|$ – Cardinality of $\Lambda(c)$
- sR_c – Source router of channel c
- sR_f – Source router of end-to-end communication flow f
- nF_f – Number of flits that flow f contributes to the contention on channel c
- m – Number of flits a packet contains
- $minHop$ – Minimum hop count
- $dictHop$ – A dictionary storing flows and their corresponding hop counts and flow is the key

- $dictHop[f]$ – Hop count from the source router of flow f to the source router of channel c: $|C_{SR_f \to SR_c}|$
- $sumProp$ – Sum of flows' effective proportions that use channel c

This refinement procedure is performed only for channels that are used by more than one end-to-end communication flow. It mainly does three jobs. First, for each flow f, it computes the hop count from its source router to the source router of the channel c and stores the calculated result into a dictionary structure ($Lines$ 6 – 9). After that, it finds the minimum value from these calculated hop counts ($Line$ 10). Its second job is to compute the sum of these flows' effective proportions ($Lines$ 11 – 16). For each flow f, the number of flits it contributes to the contention on channel c is estimated as follows:

$$nF_f = m - |dictHop[f] - minHop| \tag{4.7}$$

If the computed result is negative, it means this flow's source router locates

Algorithm 3 Procedure of refining shared channel bandwidth

```
1:  procedure RefineSCB(c, Λ(c), m)
2:      if |Λ(c)| > 1 then
3:          minHop = 0, sumProp = 0
4:          Let dictHop be a new dictionary
5:          sR_c = GetSrcRouter(c)
6:          for ∀f ∈ Λ(c) do
7:              sR_f = GetSrcRouter(f)
8:              dictHop[f] = CalcHopCount(SR_c, SR_f)
9:          end for
10:         minHop = GetMinHop(dictHop)
11:         for ∀f ∈ Λ(c) do
12:             nF_f = CalcNumFlits(f)
13:             if nF_f ≥ 0 then
14:                 sumProp += nF_f / m
15:             end if
16:         end for
17:         b_S(c) = b / sumProp
18:     else
19:         b_S(c) = b
20:     end if
21: end procedure
```

too far from channel $c's$ source router. This flow does not contribute any flit

to the contention on channel c. In another word, $sumProp$ can be thought of as the effective number of flows using channel c. Finally, channel $c's$ shared bandwidth is computed at *line* 17.

We refine channel $c'_{23}s$ shared bandwidth using the proposed procedure. If m is assumed as 5, then $b_S(c_{23})$ is equal to $\frac{1}{2.4}\ flit/cycle$, which is greater than $\frac{1}{3}\ flit/cycle$.

4.1.3.3 Estimation of end-to-end communication flow latency with network contention

A flow $f_{i\to j}$ uses a set of channels $C_{R_i\to R_j}$ to deliver its packet. These channels have their own shared channel bandwidths under network contention. The time the head flit needs to traverse them is summed up channel by channel: $\sum_{\forall k\in C_{R_i\to R_j}} \frac{1}{b_S(k)}$. The bottleneck channel c_B is used to represent the channel, which has the minimum shared channel bandwidth: $\min_{\forall k\in C_{R_i\to R_j}} b_S(k)$. Overall latency of the flit pipeline is determined by the larger value of switching delay and bottleneck channel delay. As network contention of a communication round exists only in channels between routers, t_{inj} and t_{ej} are still equal to $t_{channel}$. Equation (4.5) is modified based on these changes to estimate the end-to-end communication flow latency under network contention:

$$\begin{aligned}\sigma(f_{i\to j}) = [(|C_{R_i\to R_j}|+1)\times(t_R+t_S)+\\ \sum_{\forall k\in C_{R_i\to R_j}} \frac{1}{b_S(k)} + 2\times t_{channel}]+\\ [max(t_S, \frac{1}{b_S(c_B)})\times(m-1)]\end{aligned} \tag{4.8}$$

4.1.4 Evaluation of proposed equation and refinement procedure

For the purpose of experiments and evaluation, we implemented a program named *fnoc* in C++ to compute the latency of a communication round for mesh NoCs. In the first step, it computes routes for end-to-end communication flows in the provided communication round. After that, each flow knows

which channels it passes through. And each channel knows which flows use it. Then it computes the shared channel bandwidth for each flow based on the proposed refinement procedure. In the last step, the latency of each flow could be computed using the proposed Equation (4.8). The maximum value is the latency of this communication round.

To validate the accuracy of our proposed method, we compare it with a cycle-accurate NoC simulator [152]. We extended it with fault injection. If a router suffers from faults, it cannot send and receive flits. Its directly connected neighbors know its status. If a packet cannot be further delivered, it is discarded and not counted as a successfully delivered packet. Furthermore, we assign the input buffer size to 1000 flits in order to make sure it is large enough for storing incoming flits.

We use following parameters in both programs:

- Routing delay (t_R): 2 cycles
- Switching delay (t_S): 1 cycle
- Physical channel bandwidth (b): 1 flit/cycle
- Packet size: 20 flits
- Routing function: XY and FTNF

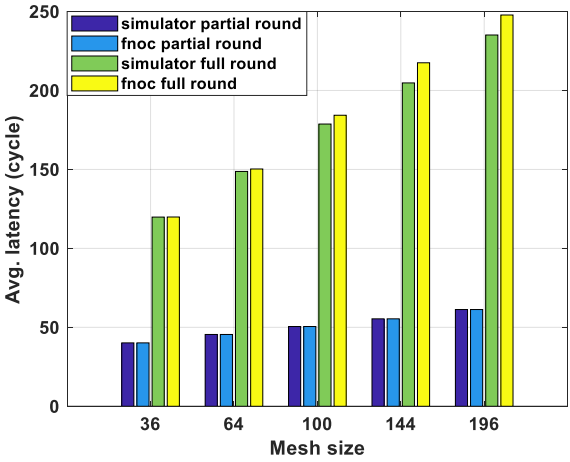
Totally, three different test cases were carried out on meshes of varying sizes: 36 (6×6), 64 (8×8), 100 (10×10), 144 (12×12), and 196 (14×14) with consideration of two different routing algorithms (XY and FTNF):

- Partial communication round containing only one flow and full communication round under fault-free condition and two routing algorithms
- Partial communication round containing only one flow and full communication round under fault condition of 10 percent faults and two routing algorithms
- Partial communication round containing only one flow and full communication round under fault condition of 20 percent faults and two routing algorithms

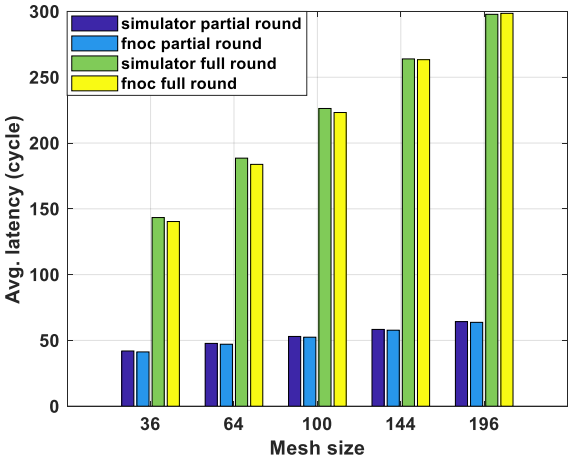
All experiments were carried out on the same host machine equipped with Intel Core Duo processor at 2.4 GHz and 12 GB RAM.

4.1.4.1 Evaluation case: fault-free

In the case of a fault-free situation, 1000 communication rounds were first generated for each mesh NoC, to be simulated with the simulator and *fnoc*.



(a) Average latency comparison under fault-free condition and XY routing



(b) Average latency comparison under fault-free condition and FTNF routing

Fig. 4.7: Average latency comparison under fault-free condition

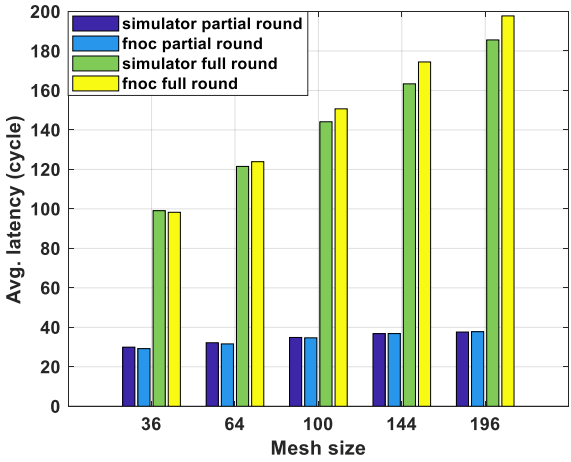
Average latencies and simulation times from both programs for different size mesh NoCs and two different routing algorithms (XY and FTNF) are listed in Figure 4.7a, Figure 4.7b and Table 4.2 respectively. As can be seen, the estimated latencies from *fnoc* for partial communication rounds are almost the same as the results from the simulator. In addition, our program's average accuracy for estimating latencies of full communication rounds under XY routing is 96.84%. In contrast, the average accuracy for computing latencies of full communication rounds under FTNF routing is 98.71%. Of particular interest is the average speedup of 53 that we achieve in comparison to the simulator.

Table 4.2: Simulation time of full round under fault-free condition

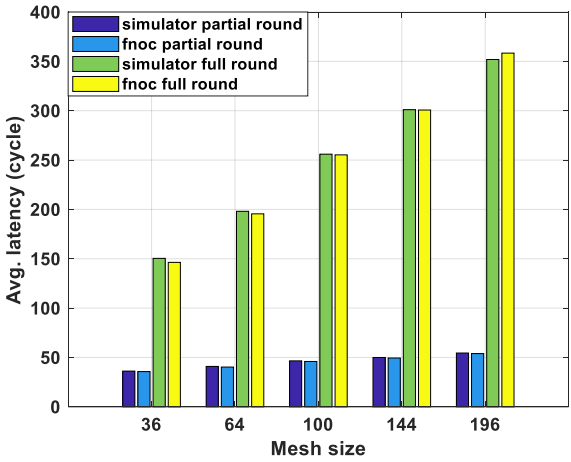
Mesh	**XY (ms)**		**FTNF (ms)**	
	simulator	**fnoc**	**simulator**	**fnoc**
36	7.54	0.12	7.83	0.14
64	15.12	0.26	16.23	0.32
100	28.32	0.53	31.21	0.58
144	44.92	0.89	46.46	0.96
196	69.25	1.42	72.35	1.51

4.1.4.2 Evaluation case: 10 percent faults

In this evaluation case, 500 different fault combinations were generated for each mesh at first. The number of faulty routers is 10 percent of the total number of routers. For each fault combination, 100 communication rounds were generated and then used in both programs. Thus, 50000 communication rounds were evaluated for each mesh. Average latencies of XY and FTNF routings and simulation time from both programs are given in Figure 4.8a, Figure 4.8b and Table 4.3. Comparing to fault-free cases, our program achieves an average speedup of 71. Furthermore, it achieves the average accuracies of 95.87% and 98.76% for full communication rounds under XY and FTNF routing algorithms respectively. For partial communication rounds, it provides results that are quite close to that as offered by the simulator.



(a) Average latency comparison under fault condition (10 percent) and XY routing



(b) Average latency comparison under fault condition (10 percent) and FTNF routing

Fig. 4.8: Average latency comparison under fault condition (10 percent)

Table 4.3: Simulation time of full round under fault condition (10 percent)

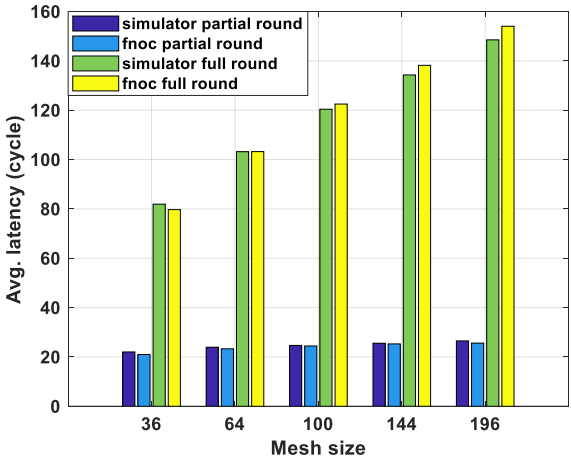
Mesh	XY (ms)		FTNF (ms)	
	simulator	fnoc	simulator	fnoc
36	5.71	0.073	6.48	0.098
64	11.26	0.15	14.71	0.24
100	20.03	0.27	26.43	0.44
144	39.04	0.43	41.72	0.71
196	51.90	0.65	65.44	1.11

4.1.4.3 Evaluation case: 20 percent faults

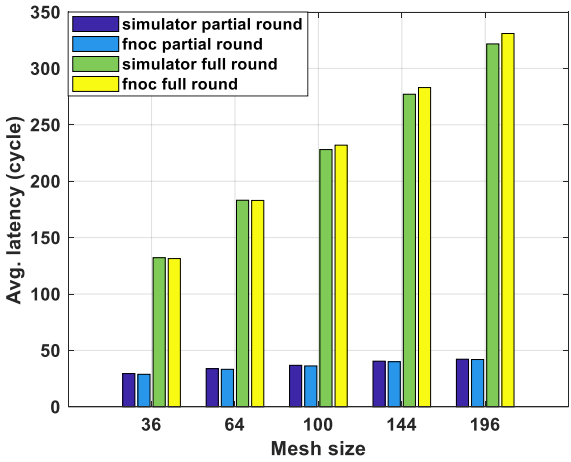
For this evaluation case, we first generated 500 random fault combinations for each mesh. The number of faulty routers is 20 percent of the total number of routers. Then 100 full communication rounds were generated and used in both programs. Therefore, we evaluated 50000 communication rounds for each mesh. Average latencies of XY and FTNF routings and simulation time from both programs are given in Figure 4.9a, Figure 4.9b and Table 4.4. As can be seen, our program obtains the average accuracies of 97.78% and 98.52% for full communication rounds under XY and FTNF routing algorithms respectively. Moreover, it achieves nearly the same results as produced by the simulator for partial communication rounds under both routing algorithms. It is worth mentioning that the average speedup of 95 that we achieve in comparison to the simulator.

Table 4.4: Simulation time of full round under fault condition (20 percent)

Mesh	XY (ms)		FTNF (ms)	
	simulator	fnoc	simulator	fnoc
36	5.21	0.041	5.72	0.075
64	10.38	0.096	13.24	0.17
100	18.57	0.17	22.57	0.29
144	28.94	0.24	39.22	0.53
196	39.52	0.37	53.27	0.76



(a) Average latency comparison under fault condition (20 percent) and XY routing



(b) Average latency comparison under fault condition (20 percent) and FTNF routing

Fig. 4.9: Average latency comparison under fault condition (20 percent)

4.2 Fault resilience

In this section, we detail another reward metric named fault resilience and a fast approach to computing it. The ratio of the successfully delivered packets to the total number of injected packets is defined as fault resilience (F) in our work. The total number of injected packets is equal to the sum of the successfully delivered packets and the dropped packets because of faults. Fault resilience is an indicator of how reliable a NoC is in terms of connected paths. Its expression is given in Equation (4.9), where:

- P_s: number of the successfully delivered packets
- P_t: number of the totally injected packets

$$F = \frac{P_s}{P_t} \tag{4.9}$$

The commonly used approach to analyzing the fault resilience of fault-tolerant routing algorithms is to use a cycle-accurate simulator. In this work, we introduce a fast approach to computing fault resilience based on channel connectivity matrix and search instead of simulations. Our proposed approach is exemplified using FTNF routing. It achieves a considerable speedup in comparison with lengthy simulations.

4.2.1 Powers of adjacency matrix

In graph theory, the adjacency matrix is one way to represent a network. It is a square matrix with rows and columns labeled by the graph's vertices. The elements of the matrix indicate whether pairs of vertices are adjacent or not [92]. Mathematically, the adjacency matrix A of a graph G having n vertices $(v_0, v_1, \ldots, v_{n-1})$ is an $n \times n$ matrix with elements $A_{i,j}$ such that

$$A_{i,j} = \begin{cases} 1, & \text{if there is an edge from vertex } v_i \text{ to vertex } v_j \\ 0 & \text{otherwise} \end{cases}$$

For example, let G_0 be the 5-node directed graph with 6 edges illustrated in Figure 4.10. Its adjacency matrix A_{G_0} is given as follows:

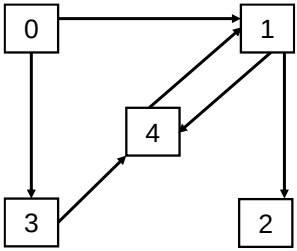


Fig. 4.10: A 5-node directed graph with 6 edges

$$A_{G_0} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

A path in a graph is an alternating sequence of vertices and edges, where for each edge $\{v_i \rightarrow v_j\}$ in the path, vertex v_i is the element before the edge, and vertex v_j is the element after the edge. The length of a path is the number of its edges [150]. E.g. in Figure 4.10, there is a path from vertex v_0 to vertex v_2 of length 2:

$$0\ \{0 \rightarrow 1\}\ 1\ \{1 \rightarrow 2\}\ 2$$

It has been proven that the ij^{th} entry of A^m gives the number of paths from the vertex v_i to the vertex v_j of length m [102, 43]. For example, the third power of the adjacency matrix of the illustrated 5-node directed graph is computed and given in Equation (4.10):

$$A^3_{G_0} = \begin{bmatrix} 0 & 2 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.10}$$

It provides the number of paths of length 3 for any two vertices of the exemplified 5-node directed graph. As can be seen, the entry at the first row and the second column of $A^3_{G_0}(0,1)$ is equal to 2, which means there are two paths of length 3 from vertex v_0 to v_1:

$$0\ \{0 \rightarrow 3\}\ 3\ \{3 \rightarrow 4\}\ 4\ \{4 \rightarrow 1\}$$
$$0\ \{0 \rightarrow 1\}\ 1\ \{1 \rightarrow 4\}\ 4\ \{4 \rightarrow 1\}$$

Connectivity matrix denoted by β of a graph is expressed by Equation (4.11):

$$\beta = \sum_{i=1}^{u_b} A^i \tag{4.11}$$

It is the sum of powers of a graph's adjacency matrix, where the upper bound is u_b and the lower bound is 1. With help of connectivity matrix, we can find the number of paths between any node pair using the number of edges from one to u_b.

4.2.2 Channel connectivity matrix

To compute fault resilience, we need to determine whether any source and destination nodes are connected or not. In Section 4.2.1, it is known that the powers of the adjacency matrix of a graph give the total number of paths between any node pair. However, we cannot use the connectivity matrix (β) directly for NoCs. The reason is that we need to take into account the turns introduced by different routing algorithms to avoid deadlocks. Such turns are not considered in the adjacency matrix.

4.2.2.1 Channel dependency matrix

As adjacency matrix does not model such restricted turns, the authors in [100, 101] take them into account by utilizing the channel dependency matrix denoted by Φ. A channel dependency matrix is a square matrix with rows and columns labeled by the channels of a NoC, with a 1 or 0 in position (i, j) according to whether a flit can be forwarded from the channel c_i to the channel c_j. That is to say, the adjacency matrix models the node connectivity

relationships of a graph. And, the dependency relationships between channels are modeled by the channel dependency matrix.

In Figure 4.11, the forbidden turns of a mesh-based NoC of size 3x3 under FTNF routing algorithm are illustrated. Its channel dependency matrix Φ_{3x3} is shown in Equation (4.12). This NoC has totally 24 channels: c_0, c_1, ..., c_{23}. As can be seen, the channel c_0 is connected to channel c_3 through router 1. Because of FTNF routing, the east to south turn should be forbidden, the value of $\Phi_{3x3}(0,3)$ is set to 0. As turns from a negative direction (S or W) to a negative direction or a positive direction (N or E) or turns from a positive direction to another positive direction are allowed, values of such entries are set to 1, such as $\Phi_{3x3}(3,11)$, $\Phi_{3x3}(3,13)$, $\Phi_{3x3}(11,14)$ and so on.

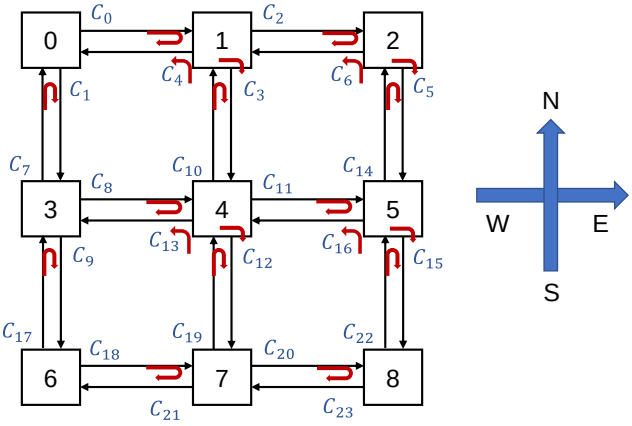


Fig. 4.11: Forbidden turns of a mesh-based NoC of size 3x3 under FTNF routing

$$\Phi_{mesh3x3} = \begin{pmatrix}
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0
\end{pmatrix} \tag{4.12}$$

Besides restricted turns, the faults of routers or channels can also be modeled in the channel dependency matrix. If a channel is faulty, all the entries from it or to it in the channel dependency matrix will be set to zero. Similarly, in case of a faulty router, it can neither send packets to its neighbors nor receive packets from its neighbors. All entries from or to its output channels are made equal to zero. Besides, all entries to its input channels need to be cleared to zero. E.g. we assume that router 1 in Figure 4.11 is faulty. Its output channels are: c_2, c_3, and c_4. The router 1 has three neighbors from directions: east, south and west. Its corresponding input channels are c_6, c_{10}, and c_0. The channel dependency matrix $\Phi_{mesh3x3}$ shown in Equation (4.12) is modified as follows with consideration of the assumed fault in router 1:

$$
\Phi_{mesh3x3} = \begin{pmatrix}
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&1&1&1&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}\\
\mathbf{0}&\mathbf{0}\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&1&1&1&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&1&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&1&1&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&1&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&1&1&1&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&1&1&1&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&1&1\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&1&1&1&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&1&1&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&1&1&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&1&0&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&0&0&0&1&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&1&1&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&1&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&\mathbf{0}&\mathbf{0}&\mathbf{0}&0&\mathbf{0}&0&0&0&\mathbf{0}&0&0&0&0&0&0&0&0&1&1&1&0&0
\end{pmatrix} \tag{4.13}
$$

4.2.2.2 Powers of channel dependency matrix

Inspired by the concept of connectivity matrix, the authors in [101] define the channel connectivity matrix (CCM) as the sum of powers of a NoC's channel dependency matrix, where the upper bound on exponents is u_b and the lower bound is 1.

$$
CCM = \sum_{i=1}^{u_b} \Phi^i \tag{4.14}
$$

For a mesh of size $n \times m$, the upper bound u_b of its CCM is $2 * (n + m) - 1$, because the i^{th} power of channel dependency matrix becomes zero when i is not smaller than the mesh's size.

The total number of connected paths between any source-destination pair with consideration of restricted turns and faulty routers can be computed using Equation (4.15), where:

$$P_{src,dest} = \sum_{O_{src}} \sum_{I_{dest}} CCM(O_{src}, I_{dest}) \tag{4.15}$$

- O_{src} denotes the output channels of a source router
- I_{dest} represents the input channels of a destination router

If the number of connected paths of a source-destination pair is greater than zero, we set its connectivity to one, otherwise, its connectivity is zero.

The matrix presented in Equation (4.16) is the channel connectivity matrix of the mesh-based NoC of size 3x3 with two faulty routers (router 4 and router 8), which is depicted in Figure 4.12. Here, the source router is 6 and the destination router is 2. The output channels of router 6 are c_{17} and c_{18}. The input channels of router 2 are c_2 and c_{14}. According to Equation (4.15), the number of connected paths of the source-destination pair (6, 2) is equal to $CCM_{mesh3x3}(17, 2) + CCM_{mesh3x3}(17, 14) + CCM_{mesh3x3}(18, 2) + CCM_{mesh3x3}(18, 14) = 1$. This unique path is illustrated in dashed line and consists of channels: c_{17}, c_7, c_0, and c_2.

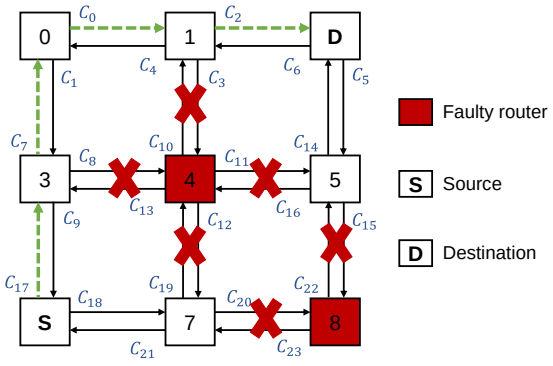


Fig. 4.12: A mesh-based NoC of size 3x3 with two faulty routers

$$CCM_{mesh3x3} = \begin{pmatrix}
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
2&0&2&0&0&0&0&2&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
3&1&3&0&0&0&0&2&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&2&1&0&0&0&0&0&0&1&0\\
3&1&4&0&1&0&0&2&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix} \tag{4.16}$$

In fact, we cannot use CCM alone to compute connectivities in the presence of multiple faults. On the one hand, $P_{src,dest}$ gives the theoretical number of the connected paths between a source-destination pair. However, the FTNF algorithm cannot use all of them. It is possible that the path chosen by the FTNF algorithm cannot reach the destination. Therefore, the connectivity may be zero even if some connected paths exist. For example, assume the source router 6 illustrated in Figure 4.12 chooses the east direction instead of north first, then the packet will arrive at router 7. Because the east and north neighbors of router 7 are both faulty, the packet cannot be forwarded anymore and will be dropped. Thus, the connectivity of the source-destination pair (6,2) is zero. On the other hand, as the mesh's size increases, the size of its corresponding channel dependency matrix grows drastically. Furthermore, the upper bound of its CCM grows as well, which results in much time needed by matrix multiplications that are applied in CCM.

4.2.3 The fast approach to computing connectivity

To avoid much time required by the CCM computation, we limit the upper bound of CCM to three, which is denoted by CCM_3. The reason for choosing three is that it reflects the implementation of the FTNF routing algorithm in the simulation. Through CCM_3, we can determine the number of connected paths for any source-destination pair, whose distance is not greater than four hops under the FTNF routing algorithm.

Obviously, we cannot only use CCM_3 to determine connectivity between any source-destination pair for large size meshes. Our approach is able to overcome this limitation. It is a combination of lookup and search, which select the same paths as specified by the FTNF routing algorithm. CCM_3 is used as a dictionary in lookup part. The purpose of the search is to find intermediate points that are along the path specified by the FTNF algorithm for a source-destination pair.

The algorithmic description of our proposed approach is given in Algorithm 4, where:

- *src*: source router of a source-destination pair
- *curr*: current router that a source-destination pair chooses. Initially, *curr* is set to src
- *dest*: destination router of a source-destination pair

It works in a recursive manner. It first checks three trivial cases ($Lines\ 2-10$). For the destination to NE direction, a packet can reach its destination router only through its west or south input channels. If the hop count between any source-destination pair is smaller than four, its connectivity can be determined by looking up CCM_3 ($Lines\ 20-22$). For other cases, we figure out the connectivity by means of intermediate points ($Lines\ 23-25$). The procedure of finding intermediate points is similar to the routing itself. It depends on the direction and is aligned with our implementation of the FTNF routing algorithm.

In Algorithm 5, the description about finding intermediate points for the destination to NE is listed. When Δ_x is equal to Δ_y, the desired intermediate point is the router, which locates at the NE direction and its offset to the current router is $(2, 2)$. In case that it is reachable, it is returned as the source router in the next round ($Lines\ 3-7$). Otherwise, the potential intermediate points need to be prepared in the explicit order based on their offsets to the current source router, which are: $(2, 1)$, $(1, 2)$, $(2, 0)$, $(0, 2)$. We loop over the potential intermediate points. When one of them is reachable, it will be returned. If all of

Algorithm 4 Procedure of finding the connected path of a source-destination pair

```
1: procedure findPath(src, curr, dest, CCM3)
2:     if curr = InvalidNode then
3:         return 0
4:     end if
5:     if getStatus(curr) = 0 || getStatus(dest) = 0 then
6:         return 0
7:     end if
8:     if curr = dest then
9:         return 1
10:    end if
11:    direction ← getDestToSrcDirection(curr, dest)
12:    Δx ← |dest.x − curr.x|
13:    Δy ← |dest.y − curr.y|
14:    if (direction = E || direction = W) & Δx = 1 then
15:        return 1
16:    end if
17:    if (direction = N || direction = S) & Δy = 1 then
18:        return 1
19:    end if
20:    if direction = NE & (Δx < 3 & Δy < 3) then
21:        return lookUp(curr, dest, direction, CCM3)
22:    end if
23:    intermediatePoint ←
24:    findIP(src, curr, dest, direction, Δx, Δy, CCM3)
25:    return findPath(src, intermediatePoint, dest, CCM3)
26: end procedure
```

them cannot be reachable, it implies a packet cannot proceed to its destination, and an invalid node is returned (*Lines* 8 − 14). When Δ_x is not equal to Δ_y, the first step is to compute the desired intermediate point, which locates at the N or E direction to the current router with a hop count of d that is the absolute value of difference between Δ_x and Δ_y (*Lines* 16 − 24). In case that the connectivity to the computed desired intermediate point does not exist, then we find the next reachable intermediate point towards the destination direction step by step (*Lines* 25 − 28).

Figure 4.13 illustrates how to compute connectivity based on our proposed approach. The current source router is 25 and destination router is 5. Routers 7, 10, 24 and 26 are faulty. The offsets between routers 25 and 5 is $(4, 4)$, therefore we need to find intermediate point to compute the connectivity

Algorithm 5 Procedure of finding the intermediate point: destination to NE direction

```
1: procedure findIP(src, curr, dest, direction, Δx, Δy, CCM3)
2:     if direction = NE then
3:         if Δx = Δy then
4:             p_desired ← getNeighbor(curr, NE, 2, 2)
5:             if CCM3(curr, p_desired) > 0 then
6:                 return p_desired
7:             end if
8:             Θ ← getPotentialIntermediatePoints(curr)
9:             for ∀p ∈ Θ do
10:                if CCM3(curr, p) > 0 then
11:                    return p
12:                end if
13:            end for
14:            return InvalidNode
15:        else
16:            d ← |Δx − Δy|
17:            if d > 4 then
18:                d ← 4
19:            end if
20:            if Δx > Δy then
21:                p_desired ← getNeighbor(curr, E, d)
22:            else
23:                p_desired ← getNeighbor(curr, N, d)
24:            end if
25:            if CCM3(curr, p_desired) > 0 then
26:                return p_desired
27:            end if
28:            return searchIPStepwise(curr, dest)
29:        end if
30:    else if ... then                          ▷ other direction cases
31:    end if
32: end procedure
```

between them. In the first round displayed in green dashed line, the desired intermediate point is the router 15. By looking up CCM_3, we determine that it is reachable. In the next round displayed in purple solid line, we check the connectivity between routers 15 and 5. As the their offsets are $(2, 2)$, which means their connectivity can be looked up in CCM_3. In this example, we can determine the connectivity between routers 25 and 5 only using two lookups.

Another example is depicted in Figure 4.14. The destination router 8 is located in the north-west direction of the source router 35. The x-offset between

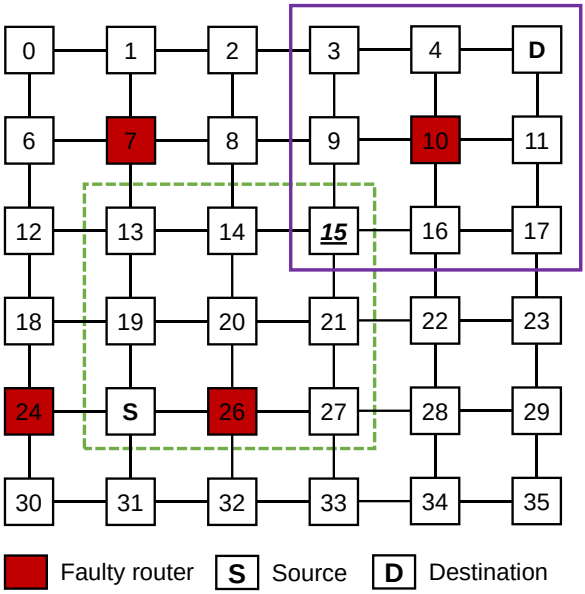


Fig. 4.13: Example of computing connectivity of a north-east source-destination pair

them is 3. Thus, the connectivity between source router 35 and router 32 is looked up in CCM_3 in the first round. As the router 32 and the destination router 8 are on the same column, the statuses of north and north-west neighbors of the router 32 are checked. If any of them is faulty, then the north neighbor of the router 32 is returned as an intermediate point. In the second round, the connectivity between router 32 and the destination router 8 needs to be computed. As can be seen, the destination router is located in the north direction of the router 32. In this situation, we first check whether the source router (src) and the current router ($curr$) are the same or not. If they are the same, it means the source-destination pair is in the first round. The statuses of the north and north-west neighbors of the source router will be checked. If both of them work well, then the north-west neighbor will be returned as an intermediate point. Otherwise, the north neighbor of the current router will be

returned. If the source router and the current router are not the same, it means we should keep forwarding towards the north. In this example, we should continue towards the north direction at router 32. As the y-offset between the destination router 8 and the current router 32 is equal to 4, their connectivity can also be looked up in CCM_3. Therefore, the connectivity of the source-destination pair can be computed through two lookups.

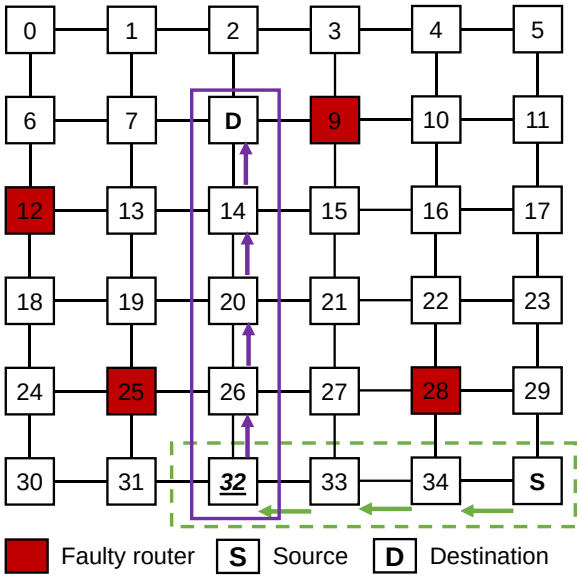


Fig. 4.14: Example of computing connectivity of a north-west source-destination pair

4.2.4 Workflow of computing fault resilience

Although we exemplified our approach to computing connectivity using the FTNF routing algorithm, other routing algorithms based on turn models can

be similarly modeled by our approach. In general, the workflow of computing fault resilience is presented as follows:

1. Generate channel dependency matrix (Φ) with consideration of the specified restricted turns and faulty routers.
2. Compute $CCM_3 = \Phi + \Phi^2 + \Phi^3$.
3. In a set of source-destination pairs, for each pair compute its connectivity using $findPath$, and sum the calculated connectivity up. Finally, the fault resilience is computed by Equation (4.9).

4.2.5 Evaluation of our approach to computing fault resilience

For the purpose of experiments and evaluation, we implemented our approach to computing fault resilience. To validate its accuracy, we compared it with a cycle-accurate NoC simulator [152], which was extended with fault injection. In addition, the FTNF routing algorithm was implemented in it. In case that a router suffers from faults, it cannot send and receive flits anymore. Its directly connected neighbors know its status. If a packet cannot be further delivered, it is dropped and not counted as a successfully delivered packet. We compared five different meshes whose sizes are: 36 (6×6), 64 (8×8), 100 (10×10), 144 (12×12) and 196 (14×14).

10000 fault combinations were generated for each mesh first. In each fault combination, the probability of all routers being faulty is the same. They were used in both programs. We carried out simulations over 20000 cycles for each fault combination. The number of faulty routers is in a range from one router to 30 percent of the total number of routers on an evaluated mesh. Our program uses the same source-destination pairs as generated by the simulator for each fault combination. All experiments were carried out on the same simulation host machine equipped with Intel Core Duo processor at 2.4 GHz and 12 GB RAM.

Table 4.5 and Table 4.6 display the results of average fault resilience and total execution time for different size meshes. As can be seen, the average fault resiliences from our approach based on CCM_3 and the approach based on CCM are the same as that from the simulator. Of particular interest are the average speedups of 99 and 71 that our approach achieves in comparison to the simulator and the approach based on CCM respectively.

Table 4.5: Average fault resilience for different size meshes

Mesh	Fault resilience		
	$simulator$	CCM	CCM_3
36	0.832	0.832	0.832
64	0.807	0.807	0.807
100	0.759	0.759	0.759
144	0.733	0.733	0.733
196	0.699	0.699	0.699

Table 4.6: Total execution time for different size meshes

Mesh	Execution time (hours)		
	$simulator$	CCM	CCM_3
36	1.2545	0.0526	0.0083
64	2.7845	0.3592	0.0224
100	4.4488	1.8647	0.0473
144	6.6393	9.4698	0.0903
196	8.9576	32.9658	0.1741

4.3 Machine learning based approach to computing performance metrics

Machine learning techniques become increasingly popular. One of the most important advantages of machine learning is to deal with complex problems. In some applications such as facial recognition and game artificial intelligence, machine learning techniques may perform close to or even better than human beings [148]. Essentially, we can use machine learning techniques to identify and exploit hidden patterns in training data. The learned patterns are applied to unknown data [20]. There are three major classes of machine learning techniques: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the training dataset is a collection of labeled pairs $(x_i, y_i)_{i=1}^{N}$, where N denotes the total number of training samples, x_i is called a feature vector and y_i is the label of x_i. The goal of a supervised learning algorithm is to use the training dataset to find a function that determines which label should be given to a new feature vector. Supervised learning is divided into two major application types: classification and regression [130].

In classification problems, the goal is to predict a class label, which is a discrete value. The spam filter is a good example of classification. Its input is an email, and its output is whether the provided email is spam or not. Regression tasks aim to predict a continuous number. E.g. predicting the price of a car given a set of features such as age, brand, mileage, etc [55].

Specifically, machine learning techniques have been researched for networking in a variety of aspects such as QoS prediction, routing, congestion control, resource management, etc. For example, authors in [135] introduced a throughput prediction system to help with bitrate selection and adaptation in HTTP adaptive streaming clients. An algorithm based on reinforcement learning was proposed in [151] to cope with end-to-end congestion on a multiuser network according to the network state. Q-learning is the widely-used model-free reinforcement learning algorithm, which learns a policy telling an agent to take suitable action under different situations. It can be found broadly in routing applications [28, 84, 21]. Recently, Q-learning has been applied in more and more applications related to resource management [96, 138, 146].

Besides, machine learning techniques have been applied in the different aspects of NoCs, e.g. predicting traffic flow latency, reducing power consumption, and exploring design space. In [116], the authors developed a framework based on the kernel-based support vector regression to predict the channel average waiting time and the traffic flow latency. Usually, NoC latency models are based on the classical queueing theory. To model a buffer in a router as an $M/M/1$, $M/G/1/N$, or $G/G/1$ queue, the following assumptions need to be made: 1) packet service time in the router is exponentially distributed, 2) packet length satisfies an exponential distribution. However, their proposed framework does not need to consider such assumptions as they directly gather related information utilizing simulation as training datasets. In [38], to reduce both static and dynamic power consumption by power-gating the links and routers at low network utilization in NoCs, machine learning-enabled sleepy storage links and routers were proposed. They used the decision tree technique to make two predictions: predicting the link utilization to find out when to power-gate the channels and routers and predicting traffic load for changing the direction of the link. In [36], authors used a machine learning approach to intelligently explore the design space to optimize the placement of planar and vertical communication links. On average, the optimization achieved by them shows 35% energy-delay-product improvement. In this section, we propose a machine learning based approach to predict performance metrics for NoCs.

4.3.1 Overview of the machine learning based methodology

With consideration of faulty routers and other simulation parameters: packet injection rates, buffer depths, and routing algorithms, it is a challenging task to compute fault resilience and communication time in an analytical manner. The definition of fault resilience in this section is the same as that defined in Section 4.2. However, the communication time mentioned in this section is slightly different from that in Section 4.1.1. Here, we do not take into account communication rounds. In contrast, we first configure the simulator with some parameters. Then let it run until the specified number of packets are successfully delivered. The simulation cycle, at which the number of successfully delivered packets reaches to or exceeds the specified number, is defined as the communication time. Its expression is in Equation (4.17), where:

- C: configuration parameters
- $\delta(t, p)$: the number of successfully delivered packets at simulation time t under the specified packet injection rate (p), which is stored in C
- N: the specified number of packets

$$\delta(CT, C) >= N \wedge \ \delta(CT - 1, C) < N \tag{4.17}$$

In our work, we leverage machine learning techniques to predict fault resilience and communication time of NoCs under the mentioned parameters. Figure 4.15 depicts an overview of our used methodology. It consists of three parts: datasets generation, finding suitable models as well as evaluating them, and applying them to compute performance metrics for performability analysis. Datasets are collected simulation results based on different configuration parameters. We use a cycle-accurate NoC simulator named Noxim for simulations [22]. Parameters such as mesh size, buffer depth, packet injection rate, traffic pattern, routing algorithms etc. can be configured. We split a dataset into two sets: a training dataset with 80% of the total samples and a test dataset with 20% of the total samples. We use a training dataset to select a regression model, and the corresponding test dataset is used to evaluate the selected model.

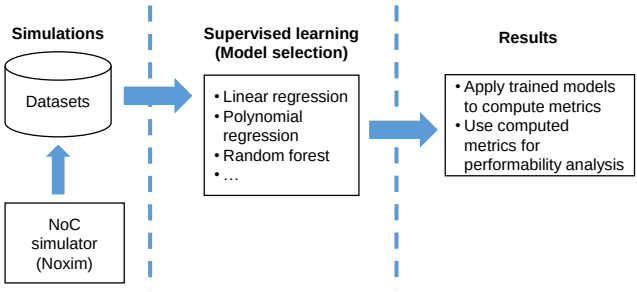


Fig. 4.15: Overview of the methodology to predict NoC's performance

4.3.2 Datasets generation workflow

To generate the datasets for training, we extended simulator with fault injection, and the FTNF routing algorithm was implemented in it. Figure 4.16 shows the workflow of generating training data. It consists of following components: configuration file, datasets generator, NoC simulator and files storing generated datasets.

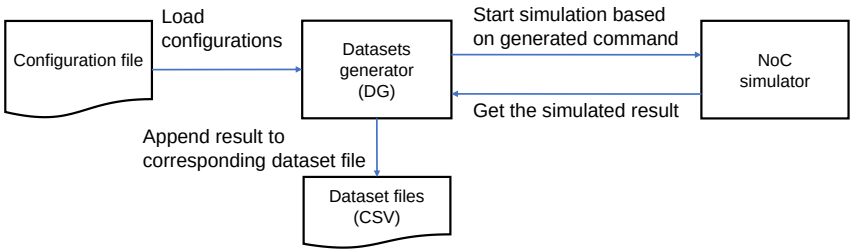


Fig. 4.16: Workflow of datasets generation

We detail how our program named datasets generator (DG) works in Algorithm 6, where:

- C: configuration parameters

- R: set of routing algorithms (XY and FTNF) specified in configuration
- P: set of packet injection rates specified in configuration
- T: set of performance metrics specified in configuration
- S: set of valid states from its Markov model
- Θ: set of generated fault combinations

Algorithm 6 Procedure of generating datasets

```
1: procedure DatasetsGenerator(configFile)
2:     C ← loadConfiguration(configFile)
3:     R ← getRoutings(C)
4:     P ← getPacketInjectionRates(C)
5:     T ← getMetrics(C)
6:     S ← generateStates(C)
7:     for each s ∈ S do
8:         Θ ← generateFaultCombs(s, 100)
9:         for each θ ∈ Θ do
10:            injectFaults(θ)
11:            for each r ∈ R do
12:                for each p ∈ P do
13:                    for each t ∈ T do
14:                        command ←
15:                        generateCommand(r, p, t)
16:                        startSimulation(command)
17:                        result ← readResult()
18:                        appendResult(result, θ, p, r, t)
19:                    end for
20:                end for
21:            end for
22:        end for
23:    end for
24: end procedure
```

The simulation parameters are stored in a configuration file. DG first loads it and parses the specified simulation parameters. Then, DG generates a set of valid states based on the utilized Markov model (*Lines* 2 – 6). In this way, the generated datasets will be evenly distributed, as all valid states of the applied Markov model are equally processed. For each state, DG generates a set of fault combinations. In our work, the maximum number of fault combinations generated for each state is 100 (*Line* 8). For each fault combination, different simulation commands are generated based on injection rates, routing algorithms and performance metrics. Before starting the simulation, the generated

fault combination is injected into simulator first. Then the DG waits until the simulation finishes. Once the simulation result is available, it appends the result to the corresponding dataset file (*Lines* 9 – 18).

In this work, we use mesh 8×8 as an example to demonstrate our methodology. The maximum number of faulty routers is set to 7. Research papers [111, 10, 137] have shown that the NoC will eventually saturate with more and more packets injected into the network. After a NoC is saturated, it behaves abnormally. It is not worth studying its performance anymore. In order to generate the valid datasets from simulation, we need to find out the saturating packet injection rate first. Our procedure for determining the saturating packet injection rate contains two steps: 1) obtain a set of global average delays under different packet injection rates, 2) find out the packet injection rate at which the global average delay increases drastically. After carrying out experiments, we find the saturating packet injection rates for XY and FTNF routings on the mesh 8×8 in the presence of 7 faults are 0.03 and 0.013 packet/cycle/node illustrated in Figure 4.17 with following simulation parameters:

- Packet size: 5 flits
- Buffer depth: 6 flits
- Traffic pattern: uniform random

The above listed simulation parameters are also used for generating datasets, and we choose the packet injection rate in a range from 0.002 to 0.012 packet/cycle/node, as mesh 8×8 will not become saturated under both XY and FTNF algorithms in this range. For the metric of communication time defined by Equation (4.1.1), the specified number of packets that need to be successfully delivered is set to 5000. For the metric of fault resilience, the simulation time is set to 10000 cycles. For each metric under each routing, a dataset is generated. Finally, we obtained four datasets that are publicly available. Two datasets contain information about communication time under XY and FTNF routings respectively. Besides, another two datasets consist of information about fault resilience under XY and FTNF routings respectively. [1]

[1] https://github.com/jiehou/noc_data

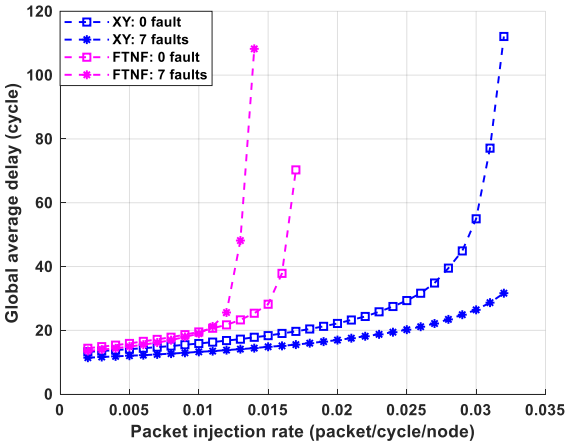


Fig. 4.17: Global average delays under different packet injection rates for mesh 8x8 with uniform random traffic

4.3.3 Model selection

For each metric under each routing, a regression model needs to be selected. Three commonly used models are chosen as candidates: linear regression, polynomial regression, and random forest regressor. In each generated dataset, the number of features is 65: packet injection rate and statuses of 64 routers. In case that the degree of the polynomial regression is set to 2, the total number of transformed features is 2211. If the degree of the polynomial regression is set to 3, the total number of transformed features is 50116. It takes too much time to train such a model. Finally, we choose the polynomial regression of degree 2. Random forest regressor is an ensemble learning method. It consists of a certain number of decision trees and uses averaging to improve the predictive accuracy and control overfitting. The number of decision trees is set to 100 after conducting experiments.

We use the k-fold cross-validation technique to choose the regression model that fits data well. The algorithmic description of model selection is listed in Algorithm 7, where:

- M: set of candidate regression models
- D: a training dataset
- k: total number of cross-validation rounds
- Π: set of equally-sized subsets
- Δ: array stores average cross-validation score of each model

Algorithm 7 Procedure of model selection

```
1:  procedure ModelSelection(M, D, k)
2:      Π ← partitionData(D, k)                          ▷ Π = D_1, ..., D_k
3:      Δ ←Ø
4:      for each m ∈ M do
5:          score ←Ø
6:          for i = 1...k do
7:              trainModel(m, D\Π[i])
8:              score[i] ← computeValidationScore(m, Π[i])
9:          end for
10:         Δ[m] ← (1/k) Σ_{i=1}^{k} score[i]
11:     end for
12:     return selectModel(Δ[m])
13: end procedure
```

First a training dataset is equally split into k subsets (*Line* 2). Then we perform totally k validation rounds for each candidate model. In each round, one of the k subsets is used as the validation dataset and other $(k-1)$ subsets are for training the candidate model. The validation score used in our work is root mean squared error (RMSE). Finally, the validation scores of the k rounds are averaged and stored for the corresponding model (*Lines* $4-11$). The model that achieves best average cross-validation score is selected (*Line* 12).

In the field of applied machine learning, k is normally selected as 5 or 10 in the k-fold cross-validation. We set k to 5 in this work. The 5-fold cross-validation is shown in Figure 4.18. We use a training dataset to perform model selection. 80% of the data from a training dataset is for training in each validation round, and 20% of the data from a training dataset is for validation. Finally, the validation scores of the 5 rounds are averaged.

The cross-validation scores and relative errors are listed in Table 4.7 and Table 4.8. The relative error is defined as the ratio of the value of RMSE to the average value of its corresponding training dataset. Based on them, we select random forest regressor for predicting communication time and fault

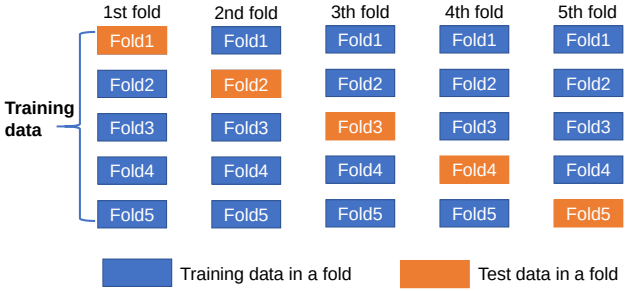


Fig. 4.18: Example of 5-fold cross-validation

Table 4.7: Cross-validation scores of regression models for communication time

Regression models	**RMSE**		**Relative error**	
	XY	$FTNF$	XY	$FTNF$
Linear regression	7838.26	5932.08	32.74%	31.21%
Polynomial regression (degree=2)	3714.69	2797.63	15.52%	14.72%
Random forest (num_trees=100)	2800.98	1389.24	11.70%	7.31%

Table 4.8: Cross-validation scores of regression models for fault resilience

Regression models	**RMSE**		**Relative error**	
	XY	$FTNF$	XY	$FTNF$
Linear regression	0.035	0.039	4.59%	4.14%
Polynomial regression (degree=2)	0.009	0.013	1.18%	1.38%
Random forest (num_trees=100)	0.007	0.004	0.92%	0.42%

resilience under XY and FTNF routing algorithms, because it achieves best scores for both performance metrics.

4.3.4 Model evaluation

In this section, we detail how the selected model works on the test datasets for communication time and fault resilience. Initially, the datasets from simulations are randomly shuffled and split into training and test datasets. The training datasets are used to select models. The test datasets are saved and untouched. First we train the random forest regressor with 100 decision trees using the whole training dataset, then we evaluate the trained model on the corresponding untouched test dataset. In addition to the RMSE score, we define the accuracy score as follows:

$$Accuracy = 1 - \frac{1}{m}\sum_{i=1}^{m}(|\hat{y}_i - y_i|/y_i) \tag{4.18}$$

where $\hat{y}_i$ denotes the predicted value of the i^{th} data from a test dataset, y_i is the label value of the i^{th} data from a test dataset, and m represents the size of a test dataset.

Table 4.9: Evaluation scores of models for communication time

Routing	RMSE	Accuracy	Training time (sec)
XY	2779.39	93.32%	86.07
FTNF	1319.93	95.93%	92.87

Table 4.10: Evaluation scores of models for fault resilience

Routing	RMSE	Accuracy	Training time (sec)
XY	0.0069	99.26%	72.68
FTNF	0.0038	99.70%	74.89

Table 4.9 and Table 4.10 list the scores of trained random forest models for communication time and fault resilience under XY and FTNF algorithms. As can be seen, the accuracies of the trained models for predicting communication times under XY and FTNF routing algorithms are 93.32% and 95.93%.

For fault resilience, the accuracies of the trained models for XY and FTNF algorithms are 99.26% and 99.70%.

The total time to get four datasets from simulations is 4245 minutes. The average time to get a performance metric from simulator is 566 ms. With consideration of training time, the average time to get a performance metric under the proposed machine learning approach is 3.7 ms, which is 153× faster than simulator. However, the model can be trained only once and saved for future use. Without consideration of training time, the average time to compute a performance metric under the machine learning approach is 0.0827 ms. In this case, we get an average speedup of 6844× comparing to simulator in terms of execution time.

4.4 Multi-threading based tool to obtaining performance metrics

From Chapter 3, it is known that there are many states in a Markov model of a NoC system. To perform the performability evaluation of such a system, a performance metric needs to be computed for each valid state. The simple and intuitive way is to compute performance metrics state by state. It will take much time to compute performance metrics serially. Multi-threaded programming is widely used to improve throughput or to improve responsiveness because it enables multiple calculations in parallel [59]. Authors in [121, 45] have utilized multi-threading techniques to accelerate the NoC simulations. In this section, we propose a multi-threading based tool to obtain performance metrics for valid states. Besides, we use a 3D mesh-based NoC of size 4x4x3 illustrated in Section 3.4 to demonstrate our proposed tool.

We use a simulation-based method to obtain precise fault resilience and communication time under consideration of faulty links or TSVs and other simulation parameters such as packet injection rate, buffer depths, and routing algorithms. We extend a cycle-accurate NoC simulator [22] to support simulations of 3D mesh-based NoCs. Parameters such as mesh size, TSV positions, buffer depth, packet injection rate, traffic pattern can be configured. Figure 4.19 depicts the structure of our tool to obtain performance metrics of valid states from the utilized Markov model, which was described in Section 3.4. In this tool, some concurrent threads are used. The number of utilized concurrent threads depends on the computer hardware deployed for simulation. E.g. on an experimental environment with 32 cores, in case that the total number of valid

states is 600 and the deployed number of threads is 30, each thread processes only 20 states.

Each concurrent thread is responsible for computing performance metrics for the states, whose range is specified when a thread is created. The number of states that each thread needs to process is evenly distributed. Finally, the results from all utilized concurrent threads are saved in a file.

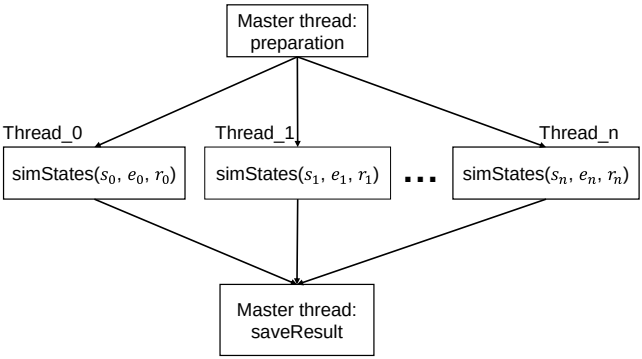


Fig. 4.19: Structure of our proposed tool to obtain performance metrics in parallel

We detail the task that each concurrent thread performs in Algorithm 8. For each state, it generates a set of fault combinations. In this work, the maximum number of fault combinations generated for each state is 1000. For each fault combination, a simulation command is generated based on the provided simulation parameters such as performance metric, routing algorithm, buffer length, packet injection rate. For the metric of communication time defined by Equation (4.17), we set the specified number of packets that need to be successfully delivered to 5000. For the metric of fault resilience, the simulation time is assigned to 10000 cycles. Before starting the simulation, the generated fault combination is injected into simulator. Once the simulation result is available, it is accumulated. In the end, the averaged value of the specified performance metric is computed and appended to the result vector.

The average time to obtain a performance metric under our proposed parallel manner is 65 ms. The utilized number of threads is 16. Our host machine is equipped with Intel Xeon CPU at 3.2 GHz, and it totally has 32 cores.

Algorithm 8 Task that a concurrent thread performs

```
1: procedure simState(s_i, e_i, r_i)
2:     for each s ∈ {s_i, e_i} do
3:         Θ ← generateFaultCombs(s, nCombs)
4:         sumRes ← 0.0
5:         numLoops ← 0
6:         for each θ ∈ Θ do
7:             injectFaults(θ)
8:             command ← createCommand(param)
9:             startSimulation(command)
10:            sum += readResult(param)
11:            numLoops++
12:        end for
13:    end for
14:    r_i.push_back(sumRes/numLoops)
15: end procedure
```

However, the average time to obtain a performance metric without the help of the multi-threading technique is 944 ms. As can be seen, our proposed method to compute performance metrics achieves a speedup of 14.5.

Chapter 5
Case studies

So far, we presented how to perform Markov modeling of an evaluated NoC based on the utilized fault model in Chapter 3. In Chapter 4, we discussed two performance metrics and different approaches to computing them efficiently. In this chapter, we present how to evaluate the performability of NoCs based on our proposed Markov modeling and performance metrics. We discuss five performability evaluation case studies for various NoC architectures, fault models, performance metrics, and approaches to obtaining them as illustrated in Figure 5.1. Which architecture, fault model, performance metric, and its computation approach apply in each evaluation case is stated at the beginning of the individual sections.

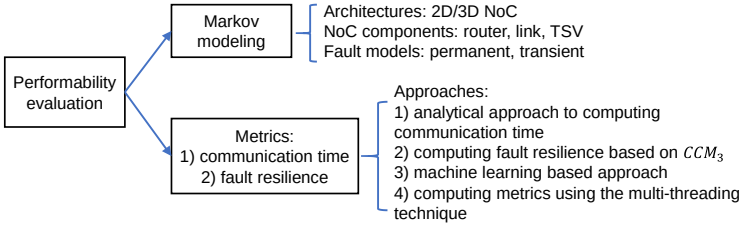


Fig. 5.1: Aspects of performability evaluation

5.1 Evaluation preliminary

5.1.1 Monte Carlo method

From Section 3.1, it is known that the utilized generic Markov state can be viewed as an array of length (z(K+2)-1). The number of fault combinations in a

state s_{eval} is equal to $\prod_{i=0}^{(z(K+2)-2)} \binom{s_{init}(i)}{s_{init}(i)-s_{eval}(i)}$, where s_{init} represents the initial state that store the number of operational routers from each router group in each horizontal layer, the number of operational links in each horizontal layer, and the number of functioning TSVs in each interlayer. The number of combinations in some states may be huge. For example, the initial state of a 2D mesh NoC of size 14x14 based on our described Markov modeling in Section 3.2 is (4, 144, 48). The number of fault combinations in the evaluated state (3, 130, 43) is equal to $\binom{4}{(4-3)} \times \binom{144}{(144-130)} \times \binom{48}{(48-43)} = \binom{4}{1} \times \binom{144}{14} \times \binom{48}{5} = 6.743e+25$. Additionally, each combination affects NoC performance differently. Therefore, a performance metric needs to be computed for each combination, which results in a huge amount of computation. Such an approach is time consuming. The Monte Carlo method is used for reducing computation time. It works as follows:

1. Randomly sample a new combination of faulty components and inject it into the evaluated network.
2. Compute the performance metric based on the combination.
3. Average all the computed performance metrics and compare the new average value with previous one. If the difference is smaller than the specified precision and the cumulative number of samples is greater than the specified minimum sample size, the process will be stopped. Otherwise, repeat these three steps until the metric converges.

The specified precision can be thought of as some positive real number.

5.1.2 Normalization of communication time

In our work, we focus on two different performance metrics: communication time and fault resilience. Based on the definition of performability, a reward of zero should be assigned to failure states. For fault resilience, it is already a normalized metric, whose value is in the range from 0 to 1. Thus, averaged fault resiliences of valid states can be directly used as rewards for the corresponding states. However, communication time is not a normalized metric. Therefore, we should first normalize the averaged communication times of valid states. The normalization of communication time is performed using Equation (5.1), where:

- N: the total number of valid states

- BT: base time. It is the minimum communication time of all valid states: $min\{CT_0,\ CT_1,\ \ldots,\ CT_{N-1}\}$
- CT_i: the averaged communication time of the i-th valid state
- r_i: the normalized communication time of the i-th valid state. After normalization, it becomes the reward of the i-th valid state

$$r_i = \frac{BT}{CT_i} \tag{5.1}$$

Finally, the long-term communication time of an evaluated NoC can be computed as follows:

$$CT_{long\_term} = \frac{BT}{\Psi} \tag{5.2}$$

where Ψ is the long-term reward rate of the evaluated system.

Similarly, the transient communication time of an evaluated NoC at the time instance of t is computed using Equation (5.3):

$$CT_{transient}(t) = \frac{BT}{\Gamma(t)} \tag{5.3}$$

where $\Gamma(t)$ is the transient reward rate of the evaluated system at the time instance of t.

5.2 Performability evaluation of mesh NoCs concerning communication time

This section evaluates the long-term communication times of mesh-based NoCs of different sizes. The configuration is as follows:

- Architecture: 2D mesh-based NoC
- Fault model: routers suffer from transient faults
- Performance metric: communication time
- Approach to obtaining communication time: analytical approach introduced in Section 4.1

Totally, five different sizes mesh-based NoCs are evaluated: 36 (6×6), 64 (8×8), 100 (10×10), 144 (12×12), and 196 (14×14). Under the listed configuration, we use the Markov modeling as described in Section 3.2 in this evaluation case.

The remainder of this section is structured as follows. Section 5.2.1 introduces how to extend our in-house tool to compute communication time based on our proposed Equation (4.8). Experimental parameters for the performability evaluations are also listed in Section 5.2.1. In Section 5.2.2, we evaluate the long-term communication times of the XY routing algorithm under the fault limit of *10* percent of total routers of an evaluated mesh NoC. Long-term communication times of the XY and FTNF routing algorithms are compared in Section 5.2.3.

5.2.1 Experimental setup

The procedure of computing communication time works as follows:

1. A full communication round is generated and its traffic pattern is uniform random.
2. Its latency is computed and added up.
3. Its successfully delivered packets are accumulated. If a targeted number of successfully delivered packets is reached, then this procedure is finished. Otherwise, repeat these three steps.

The communication time is equal to the accumulated latencies.

For states that represent faults, the above procedure is repeated for all fault combinations if their number does not exceed 10000. Communication time is averaged over these repetitions and represents the state's performance metric. For states that represent more than 10000 fault combinations, their performance metrics are computed using the Monte Carlo method described in Section 5.1.1. For this case, the specified minimum sample size is 10000.

Below, we list the utilized parameters:

- Routing delay (t_R): 2 cycles
- Switching delay (t_S): 1 cycle
- Physical channel bandwidth (b): 1 flit/cycle
- Packet size: 20 flits
- Routing function: XY and FTNF
- Specified precision of Monte Carlo method: 0.001
- Targeted number of packets need to be successfully delivered: 5000

As we compare performabilities of different size meshes, for the purpose of fairness, we assume all routers have same failure and repair rates with value

0.001 h^{-1} and 0.02 h^{-1} respectively. The global repair rate is set as 0.03 h^{-1} for all meshes.

5.2.2 Analysis of long-term communication time of XY routing

Table 5.1 provides communication times that meshes with different sizes need to successfully deliver 5000 packets under the fault-free condition. As the mesh's size increases, the communication time decreases because a larger mesh needs fewer communication rounds. E.g. a mesh of size 36 needs 139 communication rounds. In contrast, a 196 node mesh needs only 26 communication rounds. The number of routers in the mesh with size of 196 is about 5.44 times larger than that of the mesh with size 36. But its communication time is not 5.44 times as fast as the mesh of size 36. The large size mesh enables more end-to-end communication flows. However, as the network size is scaled up, the average hop counts grow too, which results in increased communication latency.

Table 5.1: Communication time comparison of different size meshes for successfully delivering 5000 packets

Mesh	Communication time (cycle)
36	16319.40
64	11754.40
100	9203.85
144	7631.92
196	6218.27

A zero reward rate is assigned to each non-operational state. The base times for meshes are listed in Table 5.1. Table 5.2 illustrates their long-term performabilities. As can be seen, the long-term performability decreases as the size of a mesh increases. In other words, the large size mesh loses more performance than small ones. E.g. performability of a mesh of size 36 is 24.34% higher than that of a mesh of size 196. The reason is that as the size of mesh-based NoCs increases the long-term residing probability in valid states decreases, as displayed in Table 5.3. A large size mesh has more numbers of nodes in $Group_2$ and $Group_3$ than a small size one. Many states in the Markov model of the large size mesh move to the states in next phase with

great transition rates. That is the reason for which the large size mesh has smaller residing probability in valid states. However, from the perspective of long-term communication time, the large size mesh is still preferred.

Table 5.2: Long-term performabilities and communication times of meshes with different sizes

Mesh	Performability	Communication time (cycle)
36	0.7748	21063.88
64	0.6881	17081.40
100	0.6258	14708.51
144	0.5788	13184.74
196	0.5314	11701.89

Table 5.3: Long-term residing probabilities in valid and failure states

Mesh	Long-term residing probability	
	Valid states	Failure states
36	0.9240	0.0760
64	0.8883	0.1117
100	0.8424	0.1576
144	0.8263	0.1737
196	0.8085	0.1915

Analysis of transient performability

Figure 5.2 illustrates the transient performability of various size meshes. As can be seen, a mesh of large size needs more time to reach stable performance. E.g. a mesh with size 196 after 700 hours has a smooth line. However, a mesh of size 36 has a stable performance after 200 hours.

Analysis of break-even failure rate

Until now, we assume all routers have the same failure rates. Actually, routers on large size NoCs have higher failure rates due to higher integration density.

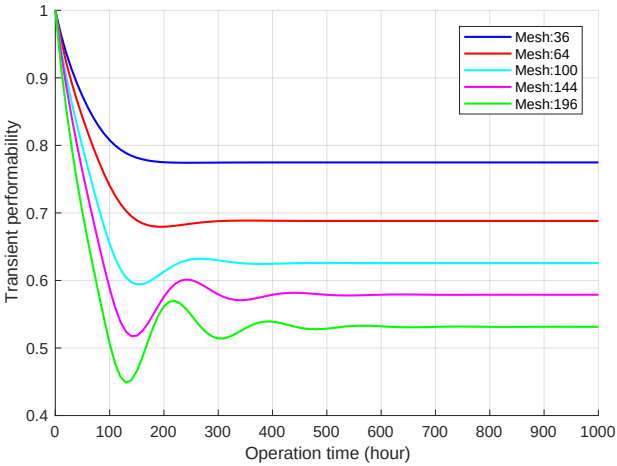


Fig. 5.2: Transient performabilities of meshes with different sizes

It is worth determining their break-even failure rates under which they can still bring net performance increase. In our work, the long-term communication time of mesh with size 36 under the failure rate 0.001 h^{-1} is used as the reference. In order to find break-even failure rate, we developed a script, which increases failure rate by an interval of 0.00001 h^{-1} at each step until the corresponding long-term communication time is greater than or equal to the reference value. Table 5.4 shows the determined break-even failure rates. As the mesh's size grows, its break-even failure rate increases as well. E.g. break-even failure rate of a mesh with size 196 is 0.00268 h^{-1} higher than that of a 64 node mesh.

Table 5.4: Break-even failure rates for meshes with different sizes

Mesh	**Break-even failure rate** (h^{-1})
64	0.00177
100	0.00273
144	0.00362
196	0.00445

5.2.3 Long-term communication time comparison of XY and FTNF routing algorithms

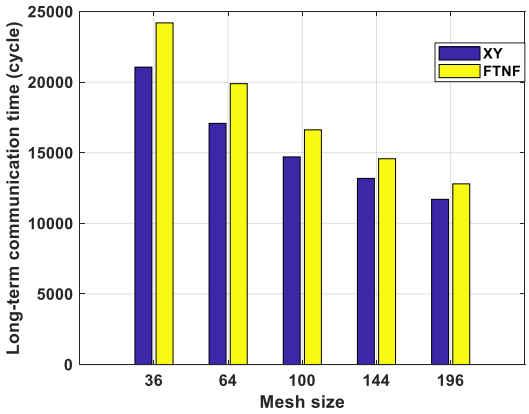


Fig. 5.3: Long-term communication time comparison of XY and FTNF routing algorithms

Figure 5.3 displays long-term communication times of the XY and FTNF routing algorithms on five different sizes mesh NoCs. As can be seen, the long-term communication times of both routing algorithms decrease as the size of the evaluated mesh increases. In addition, the long-term communication time of the XY routing algorithm is smaller than that of the FTNF routing algorithm on the same size mesh. E.g. the long-term communication time of the XY routing algorithm is 11701.89 cycles on the mesh of size 14×14, which is 1095.46 cycles less than the FTNF routing algorithm. The reason is that the introduced adaptivity by the FTNF routing results in much network contention, which makes its performance in terms of communication time worse than the XY routing algorithm. In other words, the FTNF routing needs more time for transmitting packets in each communication round. Figure 5.4 illustrates the average communication times for delivering 5000 packets under XY and FTNF algorithms on meshes 6×6 and 14×14. As can be seen,

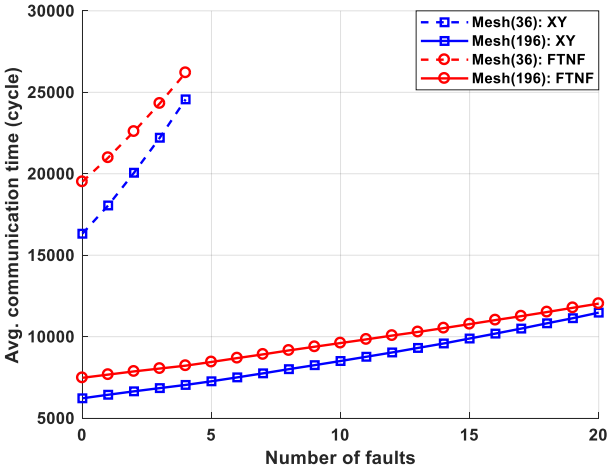


Fig. 5.4: Average communication times of XY and FTNF routing algorithms on meshes 6×6 and 8×8

the average communication time of FTNF routing on both meshes under the number of faults from 0 to 10 percent of the total number of routers is larger than that of XY routing.

5.3 Performability evaluation of mesh NoCs concerning fault resilience

We evaluate the performabilities of mesh NoCs with respect to fault resilience in this section. We utilize the following configurations:

- Architecture: 2D mesh-based NoC
- Fault model: routers suffer from transient faults
- Performance metric: fault resilience

- Approach to obtaining fault resilience: CCM_3 based approach to computing fault resilience as explained in Section 4.2

Section 5.3.1 at first lists the parameters used in conducting experiments. In Section 5.3.2, the long-term fault resiliences of XY, FTNF, and fault-tolerant XY routing algorithms on five different sizes mesh NoCs are presented and discussed. Fault-tolerant XY routing algorithm introduced in [80] is used for the purpose of comparison in this work. In Section 5.3.3, we evaluate the long-term fault resiliences of the mentioned three different routing algorithms under a high local repair rate. Finally, the transient fault resiliences of XY, FTNF, and fault-tolerant XY routing algorithms are analyzed in Section 5.3.4.

5.3.1 Experimental setup

Based on the listed configuration, the Markov modeling presented in Section 3.2 is utilized. In this section, we set the fault limit to 20 percent of the total number of routers. Table 5.5 provides the information of states for different size meshes. E.g. for mesh of size 14×14, total number of states in its Markov state space is 4105. The number of valid states is 3905.

Table 5.5: Fault limit (20 percent) and states information for meshes with different sizes

Mesh	Fault limit	No. of total states	No. of valid states
36	8	185	145
64	13	460	395
100	20	1055	955
144	29	2180	2035
196	40	4105	3905

As we compare long-term fault resiliences of meshes of different sizes, we assume all routers have same failure and local repair rates. Moreover, same global repair rate is set to all meshes. The applied traffic pattern is uniform random. Following parameters are used in our evaluation:

- Number of packets to be delivered: 10000
- Specified precision of Monte Carlo method: 0.001
- Specified sample size: 10000

- Failure rate: $0.0025h^{-1}$
- Repair rate: $0.02h^{-1}$ and $0.04h^{-1}$
- Global repair rate: $0.03h^{-1}$

5.3.2 Result analysis of long-term fault resilience

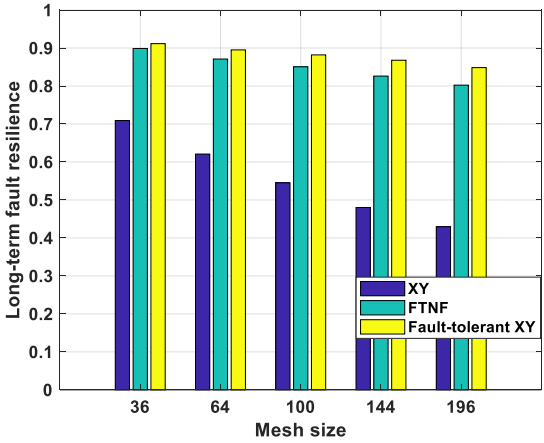


Fig. 5.5: Long-term fault resilience comparison for different size meshes

Figure 5.5 provides the experimental results for different size meshes under three routing algorithms in terms of long-term fault resilience. As size of the networks increases, the long-term fault resilience decreases. That is to say, the large size meshes lose more fault resilience than small ones from the long-term perspective. Under the assumption of the same percentage of faults, the large size meshes have a greater number of faults than small ones. In addition, communication flows in the larger size network have higher probabilities of encountering faulty routers, as they have a longer average path length.

For the same size mesh, both FTNF and fault-tolerant XY routing algorithms achieve much better long-term fault resilience than XY routing algorithm. E.g. for the mesh of size 196, the long-term fault resiliences under the FTNF and fault-tolerant XY routing algorithms are 0.8023 and 0.8484, which are 37.28% and 41.89% more resilient than that of XY routing algorithm respectively. Furthermore, the fault-tolerant XY routing algorithm can provide the best long-term fault resilience on the same mesh.

5.3.3 Result analysis of long-term fault resilience under a high local repair rate

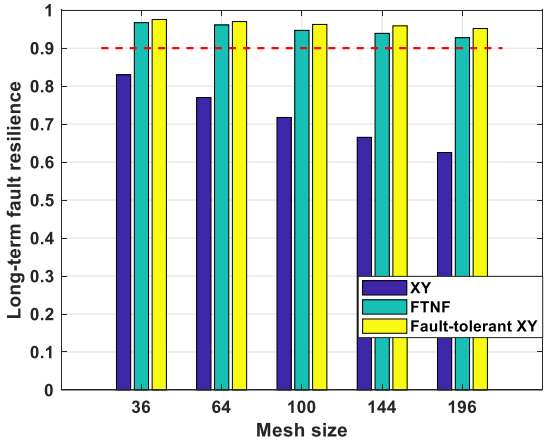


Fig. 5.6: Long-term fault resilience comparison with local repair rate of $0.04\ h^{-1}$

In this experiment, we assume local repair rate is equal to $0.04h^{-1}$. Figure 5.6 illustrates long-term fault resiliences for different size meshes under three routing algorithms. As can be seen, the long-term fault resiliences achieved

by FTNF and fault-tolerant XY algorithms are always above 0.9. As the size of meshes increases, the long-term fault resilience of FTNF or fault-tolerant XY algorithm decreases very slowly. E.g. the long-term fault resilience of FTNF on mesh 36 is 0.961, and the long-term fault resilience on mesh 196 is 0.9274. Their difference is only 0.0336. In comparison to results listed in Figure 5.5, long-term fault resiliences with a local repair rate of $0.04h^{-1}$ are better than that under a local repair rate of $0.02h^{-1}$, especially under XY routing algorithm. E.g. long-term fault resilience of XY routing on mesh 196 under a local repair rate of $0.04h^{-1}$ is 0.6251, which is 19.56% more resilient than that under a local repair rate of $0.02h^{-1}$.

5.3.4 Result analysis of transient fault resilience

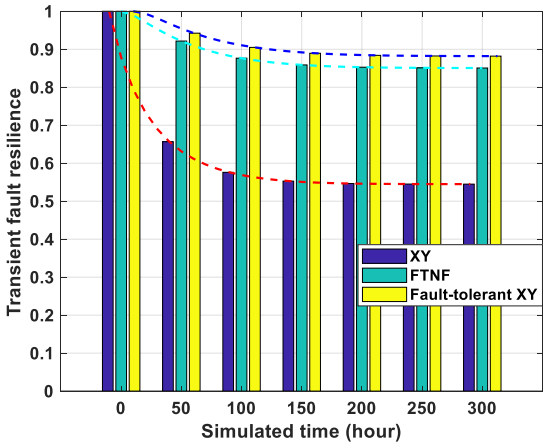


Fig. 5.7: Transient fault resiliences on the mesh NoC of size 10×10

Figure 5.7 illustrates the transient fault resiliences of the three routing algorithms on the mesh NoC of size 10×10. Utilized local repair rate is $0.02h^{-1}$.

As can be seen, as the simulated time increases, the transient fault resiliences of the three routing algorithms decrease. After about 250 hours, the transient fault resiliences does not change anymore. They become stable and are equal to the long-term fault resiliences of the corresponding routing algorithms. During the time range of 0 to 100 hours, the transient fault resiliences decrease drastically for all the three routing algorithms. However, the transient fault resiliences of FTNF and fault-tolerant XY routing decrease much slower than that of XY routing.

5.4 Machine learning based performability evaluation

In this section, we utilize a mesh NoC of size 8×8 to demonstrate our proposed machine learning based framework for performability evaluation. The utilized configuration is listed as follows:

- Fault model: routers suffer from transient faults
- Performance metrics: communication time and fault resilience
- Approach to obtaining performance metrics: machine learning based approach as described in Section 4.3

Under the listed configuration, the Markov modeling described in Section 3.2 is utilized. It is assumed that all routers have the same failure and local repair rates. Following experimental parameters are used in our evaluation:

- Packet size: 5 flits
- Buffer depth: 6 flits
- Traffic pattern: uniform random
- Failure rate: 0.0015 h^{-1}
- Global repair rate: 0.03 h^{-1}
- Specified precision of Monte Carlo method: 0.001
- Specified sample size: 10000

5.4.1 Long-term communication time under different packet injection rates

Long-term communication times from XY and FTNF algorithms under six different packet injection rates and the repair rate of 0.02 h^{-1} are illustrated in

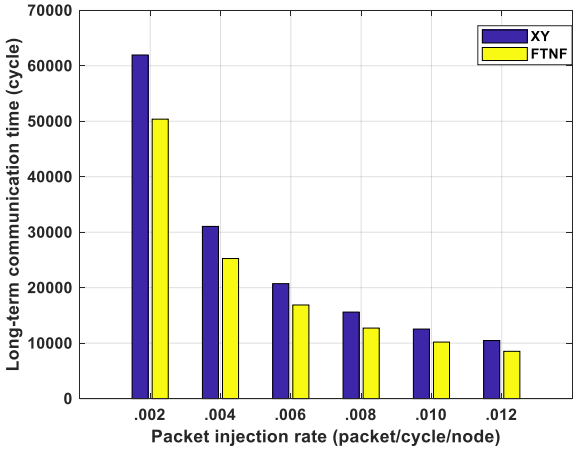


Fig. 5.8: Long-term communication times under different packet injection rates

Figure 5.8. As can be seen, the long-term communication time from the FTNF routing is less than that from the XY routing under the same injection rate. That is to say, the FTNF algorithm needs less time to successfully deliver a specified number of packets. E.g. the long-term communication time achieved by the FTNF routing at the packet injection rate of 0.01 packet/cycle/node is 2347 cycles less than that from the XY routing. When the packet injection rate is in the range from 0.002 to 0.012 packet/cycle/node, the mesh will not be fully utilized. In this case, the throughput rises as the packet injection rate increases. For the communication time, the number of packets that need to be successfully delivered is constant, therefore it decreases with increasing packet injection rate until saturation.

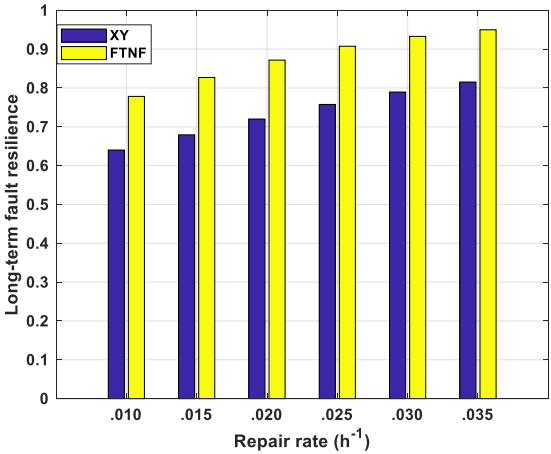


Fig. 5.9: Long-term fault resiliences under different repair rates

5.4.2 Long-term fault resilience under different repair rates

Figure 5.9 illustrates long-term fault resiliences from both XY and FTNF routing algorithms under six different repair rates. As the repair rate increases, the long-term fault resiliences from XY and FTNF routing algorithms increase as well, because a NoC has larger residing probability in valid states under a higher repair rate. E.g. long-term fault resilience from FTNF routing under the repair rate of 0.035 h^{-1} is 0.9499, which is 0.1716 higher than that under the repair rate of 0.01 h^{-1}. Under the same repair rate, the long-term fault resilience from FTNF is greater than that from XY routing. In other words, FTNF routing is more resilient than XY routing from the perspective of long-term fault resilience. E.g. under the repair rate of 0.02 h^{-1}, the long-term fault resilience from FTNF routing is 15.2% more resilient than that from XY routing.

5.5 Transient performability evaluation of 3D NoCs

In this section, we evaluate the transient performability of 3D NoCs. The configuration is as follows:

- Architecture: 3D vertically-partially-connected NoC
- Fault model: links and TSVs suffer from permanent faults
- Performance metric: communication time and fault resilience
- Approach to obtaining performance metrics: multi-threading based approach as introduced in Section 4.4

We use a 3D partially-connected mesh of size 4x4x3 as displayed in Figure 3.4 to demonstrate how to compute transient performabilities of 3D meshes. We use the following simulation parameters to obtain performance metrics:

- Packet size: 5 flits
- Buffer depth: 6 flits
- Packet injection rate: 0.01 packet/cycle/node
- Number of total TSVs: 8 (each interlayer has 4 TSVs)
- Traffic distribution: uniform random

In this study, we set the maximum allowed number of link faults in each horizontal layer to 25 percent of the total number of links in the corresponding layer. For TSVs, the maximum allowed number of faults is 50 percent of the total number of TSVs in the interlayer. Its Markov modeling has already been illustrated in Section 3.4. Based on the utilized fault limits of links and TSVs, there are 6468 states in its Markov state space. The number of valid states is 3087.

We use two different experiment settings about failure rates. The first experiment setting (exp1) assumes all links have the same failure rate of 0.00001 h^{-1} and all TSVs have same failure rates of 0.00005 h^{-1}. In the second experiment (exp2), links have a larger failure rate of 0.00002 h^{-1}. We also assume that TSVs have a larger failure rate of 0.0001 h^{-1}.

5.5.1 Implementation of 3D-FTNF routing

We target transient performabilities of two routing algorithms: elevator-first routing and 3D-FTNF routing. The elevator-first routing algorithm has been formally proven to be deadlock-free for 3D vertically-partially-connected

NoCs. There are two different cases for routing a packet toward its destination. In the first case, the source and destination nodes are in the same layer. A deterministic X-first routing algorithm is used. In the second case, the destination node is in a different layer from the source node. As each node statically knows the information of each TSVs in its layer, the source node first sends a packet to the nearest node associated with a TSV using the X-first routing algorithm. After the packet arrives at that node, it can be forwarded upwards or downwards based on whether the destination is located on an upper layer or on a lower layer. This is repeated until the packet reaches a node that is on the same layer as the destination node. After that it traverses to the destination using the X-first routing. To guarantee deadlock-freedom, each input port requires two virtual channels: ascending and descending virtual channels. If the packet's destination is on an upstairs layer, the ascending virtual channel is assigned to it. Otherwise, it gets the descending virtual channel. Once a virtual channel is assigned to a packet, it will not change during the transmission process [42].

Although the elevator-first routing algorithm can adapt to faulty TSVs, it cannot adapt to faulty links, as it uses the deterministic X-first routing on the horizontal layer. The 3D-FTNF is an extension of the elevator-first routing algorithm. The main difference is that the FTNF routing algorithm is applied in the horizontal layer. Our implementation assumes that each port has two virtual channels. Each virtual channel is associated with an input FIFO-buffer. The virtual channel allocation occurs in the processing element. When a packet is created, information about the source and destination nodes is stored in its header. Based on this information, we can assign a virtual channel for this packet. If the *z-coordinate* of the destination node is larger than that of the source node, vc_1 is assigned to this packet. Otherwise, this packet gets vc_0. Once a virtual channel is assigned to a packet, it will not change during the transmission.

The 3D-FTNF routing algorithm contains two phases: reservation phase and forwarding phase. The algorithmic description of the reservation phase is listed in Algorithm 9. In the reservation phase, the desired output direction is first computed for incoming packets from each direction and each virtual channel. After direction and virtual channel are chosen, we check the corresponding FIFO-buffer is empty or not. If it is not empty, we fetch the next available flit. If it is a header flit, the $ftnf3D$ routing function computes an output direction based on the information of the direction of input, coordinates of current and destination nodes and available TSVs. In our implementation, each router has a reservation table that stores which output direction is reserved

for which input direction and which virtual channel. If the output direction is still available, we reserve it based on the provided input direction and the allocated virtual channel.

Algorithm 9 Reservation phase

```
1: for each inDir ∈ inDirs do
2:     for each vc ∈ vcs do
3:         if !buffer[inDir][vc].isEmpty() then
4:             flit ← buffer[inDir][vc].top()
5:             if flit.type = FLIT_HEAD then
6:                 srcCoord←id2Coord(flit.srcId)          ▷ convert id to coordinate
7:                 dstCoord←id2Coord(flit.dstId)
8:                 currCoord←id2Coord(currId)
9:                 outDir←ftnf3D(srcCoord, dstCoord, currCoord, inDir)
10:                if isAvailable(outDir, vc) then
11:                    reserve(inDir, outDir, vc)
12:                end if
13:            end if
14:        end if
15:    end for
16: end for
```

We detail how the routing function of $ftnf3D$ works in Algorithm 10. As can be seen, if the coordinate of the current router is equal to the coordinate of the destination router, then the direction of $LOCAL$ is returned. In case that current and destination nodes locate on different layers, it means that the flit needs to be transmitted to an intermediate node that has a TSV. If the current node is associated with a TSV, the output direction is either up or down based on whether the destination node is on an upper layer or not. Otherwise, we find the nearest TSV to the current node, the node with the nearest TSV is an intermediate destination node. In other words, we first transmit the flit to the intermediate node using the normal FTNF routing algorithm, which is based on the negative-first turn model and was at first presented as a deadlock-free algorithm in [58]. When the flit arrives at that node, then it can be forwarded upwards or downwards.

The second phase of the 3D-FTNF routing algorithm is shown in Algorithm 11. For a flit, we first find whether it has a reserved output direction or not. In case there is no reserved output direction for this flit, this flit will be processed repeated in the coming reservation phases until it gets a reserved output direction. If the link at the output direction does not suffer from fault

Algorithm 10 ftnf3D routing function

```
1:  procedure ftnf3D(srcCoord, dstCoord, currCoord, inDir)
2:      if currCoord = dstCoord then
3:          return LOCAL
4:      end if
5:      if currCoord.z != dstCoord.z then
6:          if dstCoord.z > currCoord.z & hasUpTSV(currCoord) then
7:              return UP
8:          else if dstCoord.z < currCoord.z & hasDownTSV(currCoord) then
9:              return DOWN
10:         else
11:             dstCoord ← getNearestTSV(currCoord)
12:         end if
13:     end if
14:     return ftnf2D(srcCoord, dstCoord, currCoord, inDir)
15: end procedure
```

and the neighbor located at the desired direction has free slots, the flit can be forwarded to the next node. After that, it will be dequeued. In case that the desired link is faulty, the flit will be discarded and afterward dequeued. We assume the wormhole-switched flow control method is used in this work. After a tail flit is processed, the reserved output direction will be cleared in the reservation table.

5.5.2 Analysis of transient communication time

Figure 5.10 illustrates the transient communication times of 3D-FTNF and elevator-first routing algorithms under the two settings: *exp1* and *exp2*. The lines with squares represent results of 3D-FTNF routing, and the results of elevator-first routing are illustrated by lines with circles. As can be seen, as the simulated time increases, the communication times of both 3D-FTNF and elevator-first routing algorithms increase as well. The reason is that more and more faulty links and TSVs will appear over the simulated time. On the other hand, the gap of communication times of 3D-FTNF and elevator-first routing algorithms becomes larger as the simulated time increases. E.g. the difference between these two routings under the second experiment setting at the simulated time of 1000 hours is 417 cycles. At the simulated time of 2000 hours, the difference becomes 854 cycles. Moreover, the transient

Algorithm 11 Forwarding phase

```
1: for each inDir ∈ inDirs do
2:     for each vc ∈ vcs do
3:         if !buffer[inDir][vc].isEmpty() then
4:             flit ← buffer[inDir][vc].top()
5:             outDir ← getOutput(inDir, vc)
6:             if outDir != NOT_RESERVED then
7:                 if isLinkGood(outDir) then
8:                     if freeSlotsNeighbor(outDir, vc) then
9:                         sendFlit(flit, outDir)
10:                        buffer[inDir][vc].pop()
11:                    end if
12:                else
13:                    discard(flit)
14:                    buffer[inDir][vc].pop()
15:                end if
16:                if flit.type = FLIT_TAIL then
17:                    release(inDir, outDir, vc)
18:                end if
19:            end if
20:        end if
21:    end for
22: end for
```

communication time of elevator-first routing increases faster than that of the 3D-FTNF routing when failure rates of links and TSVs are larger. In other words, the 3D-FTNF performs better in terms of communication time in comparison with the elevator-first routing algorithm as the simulated time increases and especially when the failure rates are large.

5.5.3 Analysis of transient fault resilience

For the analysis of transient fault resilience, we assign a reward of zero to each failure state, in which a 3D NoC will not work anymore. Results of transient fault resiliences of 3D-FTNF and elevator-first routing algorithms are illustrated in Figure 5.11. It can be observed that the transient fault resiliences of both routings decrease as the simulated time goes on, because more and more faulty links and TSVs will appear over simulated time. On the other hand, the transient fault resilience of 3D-FTNF routing is always larger than that of the

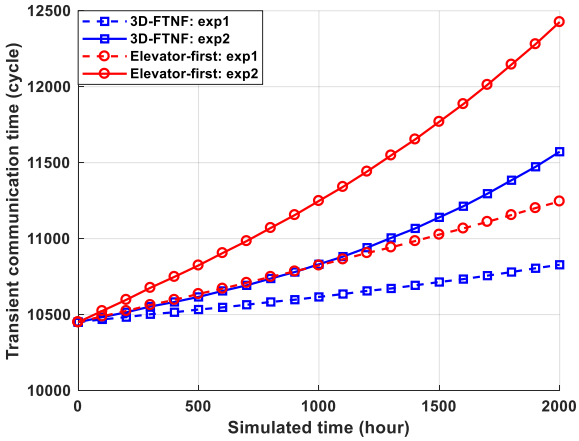


Fig. 5.10: Transient communication times of 3D-FTNF and elevator-first routing algorithms

elevator-first algorithm under these two different experiments after the evaluated 3D NoC started operation. Furthermore, the transient fault resilience of 3D-FTNF routing decreases slower than that of elevator-first routing when failure rates of links and TSVs are larger. E.g. at the simulated time of 2000 hours, the elevator-first routing loses 1.44 times as much fault resilience compared to 3D-FTNF routing. In other words, 3D-FTNF routing is more resilient than elevator-first routing from the perspective of transient fault resilience.

5.5.4 Determining maximum failure rates of links and TSVs

In this section, we determine the maximum failure rates (MFRs) of links or TSVs at which the transient fault resilience of the evaluated partially-connected 3D mesh of 4x4x3 at the simulated time of 2000 hours is smaller than 0.80. We assume that the start failure rates of links and TSVs are 0.000010 h^{-1} and 0.000050 h^{-1}. We increase both failure rates step by step in an interval

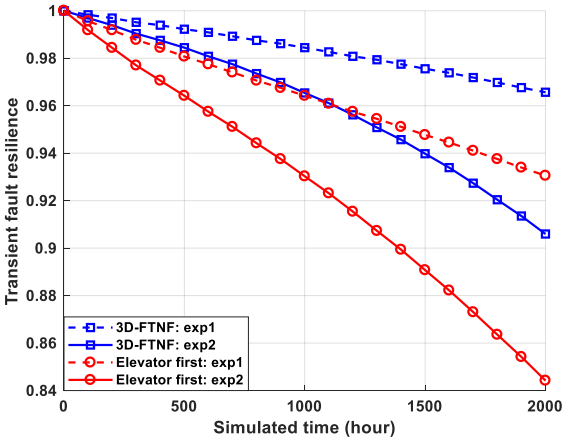


Fig. 5.11: Transient fault resiliences of 3D-FTNF and elevator-first routing algorithms

of 0.000005 h^{-1}. The obtained MFRs of links and TSVs for 3D-FTNF and elevator-first routing algorithms are listed in Table 5.6.

Table 5.6: Determined MFRs for 3D-FTNF and elevator-first routings

Routing	**MFR of links** (h^{-1})	**MFR of TSVs** (h^{-1})
3D-FTNF	0.000055	0.000095
Elevator-first	0.000035	0.000075

From the obtained results, we conclude that 3D-FTNF routing can tolerate higher failure rates of links and TSVs in comparison to the elevator-first routing under the assumption that the same fault resilience is achieved. Under the same conditions, the NoC can remain operational for a longer time using 3D-FTNF, compared to elevator-first.

5.6 Transient performability evaluation of hexagonal NoCs

Hexagonal NoC utilizes spatial redundancy by adding North-East (NE) and South-West (SW) diagonal links to the nodes of mesh topology as illustrated in Figure 2.2. These diagonal links can serve as shortcuts for packets traveling in the NE and SW directions. Thus, they help reduce the average latency. Authors in [100] proposed a fault-tolerant routing algorithm based on the negative-first turn model. We utilize this routing algorithm in our work. For hexagonal NoCs, three directions namely E, N, and NE are positive directions. Another three directions namely W, S, and SW are negative directions. In this section, we evaluate transient communication times of three different sizes hexagonal NoCs (6x6, 8x8, and 10x10). Besides, we compare transient communication times of a hexagonal NoC and a mesh NoC. The configuration is as follows:

- Architecture: 2D hexagonal NoC and 2D mesh NoC
- Fault model: routers suffer from permanent faults
- Performance metric: communication time
- Approach to obtaining performance metrics: multi-threading based approach as introduced in Section 4.4

Routers are classified into three groups based on their positions. The routers from $Group_0$ locate at the four corners. Routers that sit in outer edges belong to $Group_1$. The other ones that locate inside the mesh are classified as $Group_2$. After we have determined the state for mesh-based NoCs, the next step is to find out the transitions between possible states with consideration of failures and fault limit of routers. Our proposed model for both hexagonal NoC and mesh-based NoC under consideration of permanent faults in routers is illustrated in Figure 5.12. Here, we assume that the router fault limit n_r is at least 4. λ_0, λ_1, and λ_2 represent failure rates of routers from router groups: $Group_0$, $Group_1$, and $Group_2$ respectively. When a fault occurs in a router, a state transition happens. States from our proposed model are classified into two groups: valid states and failure states. Once the system moves into a failure state, its performance will be treated as meaningless. Besides, a failure state does not have any subsequent states in comparison to a valid state.

The utilized experimental parameters are listed as follows:

- Packet size: 5 flits
- Buffer depth: 6 flits
- Traffic pattern: uniform random
- Failure rate: 0.0001 h^{-1}

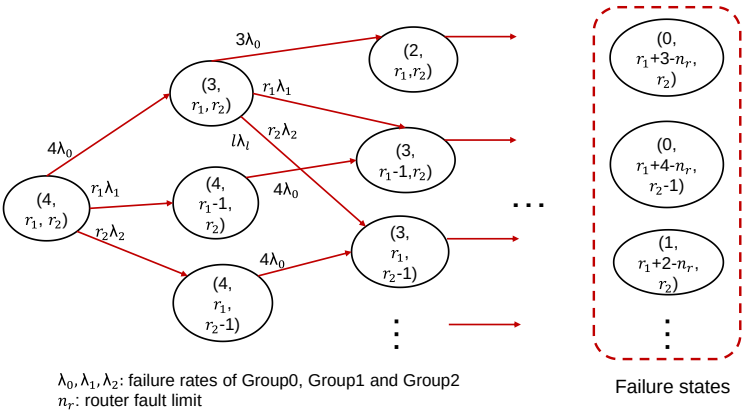


Fig. 5.12: Markov model for both mesh and hexagonal NoCs under consideration of permanent router faults

- Fault limit: 10 percent of total routers
- Packet injection rate: 0.01 packet/cycle/node
- Number of packets for communication time: 5000
- Traffic distribution: uniform random

5.6.1 Transient communication time of different sizes of hexagonal NoCs

Table 5.7 provides the derived communication times that our hexagonal NoCs with different sizes need to successfully deliver 5000 packets under failure-free conditions. As the NoC becomes larger, communication time decreases because more packets are generated at each cycle, which increases the number of communication flows. Although larger NoC leads to more average hop counts, an 8x8 hexagonal NoC, which generates 1.78 times the number of packets a 6x6 hexagonal NoC can generate at each cycle, is still 1.77 times as fast as a 6x6 hexagonal NoC.

Transient communication times of these three hexagonal NoCs are illustrated in Figure 5.13. Within a short period, a larger hexagonal NoC is still

Table 5.7: Base communication times of three different sizes hexagonal NoCs

Hexagonal NoC	Base communication time (cycles)
6x6	13905.43
8x8	7841.84
10x10	5024.24

a better choice if shorter communication time is preferred. However, since performability of a larger NoC decreases faster, the transient communication time increases faster and might exceed those of smaller hexagonal NoCs after certain time points.

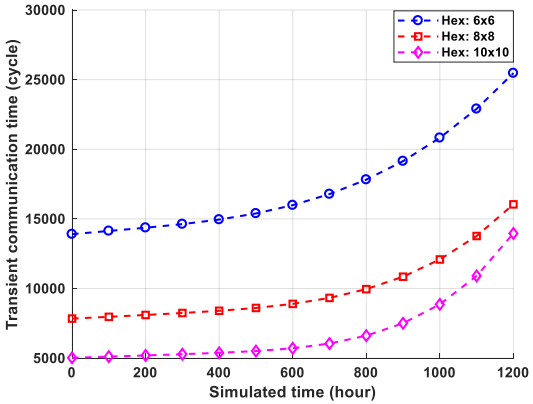


Fig. 5.13: Transient communication times of three different sizes hexagonal NoCs

5.6.2 Transient communication time comparison of a hexagonal NoC and a mesh NoC

Transient communication times of a 10x10 hexagonal NoC and a 10x10 mesh NoC are illustrated in Figure 5.14. The utilized fault-tolerant routing algorithms on both topologies are based on the negative-first turn model. Concerning shorter communication times, the 10x10 hexagonal NoC is better than the 10x10 mesh NoC. E.g. at the simulated time of 1000 hours, the transient communication time of the mesh NoC is 2769 cycles more than that of the hexagonal NoC.

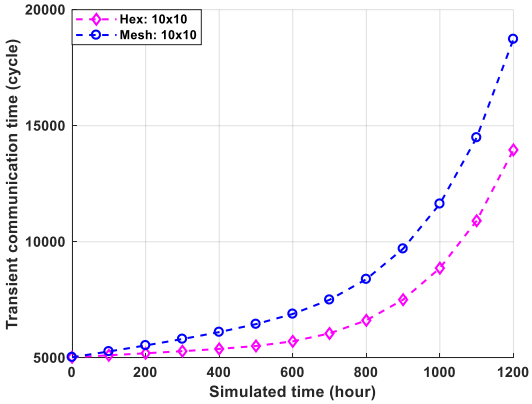


Fig. 5.14: Transient communication times of a 10x10 hexagonal NoC and a 10x10 mesh NoC

Chapter 6
Conclusion and future work

This thesis presents a methodology that allows for the performability evaluation of NoCs with respect to two different performance metrics: communication time and fault resilience. The proposed methodology is based on the Markov reward model. Moreover, the methodology is demonstrated by several performability evaluation case studies of NoCs. In this chapter, we make conclusions for each case study.

Conclusion for first case study

In the first case study, we present how to analyze the performability of mesh-based NoCs concerning communication time. Based on the evaluation results, we make the following conclusions:

- We can estimate the latency of a communication round on average 73× faster with the accuracy of 95.87% or better, allowing us to investigate performabilities for meshes of larger sizes.
- Under the assumption that failure, repair, and global repair rates are 0.001 h^{-1}, 0.02 h^{-1} and 0.03 h^{-1}, a 196 node mesh has long-term performability of 0.5314, which is 24.34% smaller than that of a 36 node mesh because the 196 node mesh has smaller long-term residing probability in valid states. In terms of transient performability, a 196 node mesh needs 500 hours longer than a 36 node mesh to reach a stable performance. With increasing size, the mesh suffers from performability loss, as the large size mesh has smaller long-term residing probabilities in the valid states. Moreover, it needs more time to reach a stable performance.
- For the same size mesh, XY routing achieves better long-term communication time despite FTNF's higher adaptivity under the assumption that the fault limit is set to 10 percent of the total number of routers of an evaluated mesh.
- The scaling towards larger NoCs is possible to bring net performance increase, but failure rate growth must be limited.

Conclusion for second case study

In the second evaluation case, we evaluated the long-term and transient fault resiliences of mesh NoCs. Fault resiliences of valid states are obtained based on our proposed approach introduced in Section 4.2. Our approach to computing fault resilience is on average 99× faster without losing accuracy comparing to cycle-accurate simulations. It allows us to investigate long-term fault resiliences for NoCs of larger sizes. We make the following conclusions based on experimental results:

- From the perspective of long-term fault resilience, the large size mesh suffers from loss. Under fault-tolerant algorithms, the long-term fault resilience decreases slowly as the size of the mesh increases.
- For the same size mesh, the long-term fault resiliences from fault-tolerant routing algorithms are better than that from the XY algorithm. Moreover, the fault-tolerant XY routing algorithm achieves the best long-term fault resilience. E.g. under the local repair rate of $0.02h^{-1}$, the long-term fault resilience of the fault-tolerant XY algorithm on mesh 196 is 4.61% and 41.89% more resilient that of FTNF and XY algorithms respectively.
- Long-term fault resiliences under a high local repair rate of $0.04h^{-1}$ are more resilient than that of a low local repair rate of $0.02h^{-1}$.
- Transient fault resiliences of FTNF and fault-tolerant XY routing algorithms decrease much slower than that of XY routing before they become stable.
- The metric of fault resilience is a measure of a NoC's ability to tolerate faults in terms of connected paths. Together with the Markov model, the long-term fault resiliences of mesh-based NoCs under different routing algorithms can be quantized. In this work, we use mesh-based NoCs to demonstrate how to compute long-term fault resilience. Our methodology can be generalized for other topologies.

Conclusion for third case study

The third evaluation case utilizes a mesh NoC of size 8x8 to illustrate our proposed machine learning based framework for performability evaluation. Through machine learning techniques, we can obtain metrics of fault resilience and communication time under consideration of a different number of faulty

routers, packet injection rates, and routing algorithms. Under FTNF routing, we can predict fault resilience with the accuracy of 99.70% and communication time with an accuracy of 95.93%. Under XY routing, the accuracies of getting fault resilience and communication time are 99.26% and 93.32%. Of particular interest, the approach to computing a performance metric employing pre-trained models has an average speedup of 6844× comparing to the Noxim simulation in terms of execution time. Based on experimental results, we conclude:

- Before the NoC becomes saturated, the long-term communication time from FTNF routing is smaller than that from XY routing. Moreover, as the packet injection rate increases, the long-term communication time from XY and FTNF routing algorithms decrease until saturation.
- As the repair rate increases, the long-term fault resiliences from XY and FTNF routing algorithms increase as well. Under the same repair rate, FTNF routing is more resilient than XY routing from the perspective of long-term fault resilience. E.g. under the repair rate of 0.02 h^{-1}, the long-term fault resilience from FTNF routing is 15.2% more resilient than that from XY routing.

Conclusion for fourth case study

The fourth evaluation case study presents how to compute transient performability of 3D NoCs with consideration of permanent faults in both links and TSVs. We use a partially-connected 3D mesh-based NoC of 4x4x3 to demonstrate the usage of our proposed framework. Our framework can as well be generalized for other architectures of 3D NoCs. Finally, we compute and compare the transient communication times and fault resiliences of 3D-FTNF and elevator-first routing algorithms with the help of the proposed framework. From evaluation results, it is concluded that 3D-FTNF routing performs better than elevator-first routing in terms of transient communication time and fault resilience, especially as the simulated time increases and the failure rates of links and TSVs are large. Moreover, 3D-FTNF routing can tolerate higher failure rates of links and TSVs comparing to the elevator-first routing.

Conclusion for fifth case study

In the fifth evaluation case study, transient communication times of three different sizes hexagonal NoCs (6x6, 8x8, and 10x10) are evaluated. A large size hexagonal is preferred concerning transient communication time. However, the transient communication time of a larger hexagonal NoC may exceed those of smaller hexagonal NoCs as its performability decreases faster. Besides, we compare a 10x10 hexagonal NoC with a 10x10 mesh NoC concerning transient communication time. On both NoCs, the utilized fault-tolerant routing algorithm is based on the negative-first turn model. Overall, adopting hexagonal topology instead of mesh topology can lead to lower latency and better performability.

6.1 Future work

In recent years, other innovative NoC architectures (e.g. wireless NoC) have been proposed and studied. A wireless NoC is a hybrid architecture consisting of both wired and wireless routers [37]. As the construction of wireless routers is more prone to faults, it is worth studying how different failure rates of wireless routers affect the performability of wireless NoC.

Another research direction is to investigate the performability of NoC-based artificial neural network (ANN) accelerators. The NoC-based neural network accelerator receives much attention recently because it can supply power efficiency, computational flexibility, and reconfigurability. Furthermore, the NoC-based design methodology decouples the ANN operation into computation and data transmission. Regarding the computation part, different ANN computing models can be performed independently of the data flow. Concerning the data transmission part, various data flows for different ANN computing modules can be efficiently processed by the NoC interconnection [26].

In this thesis, we focus on the performability evaluation of NoCs. However, there are other components such as computing cores, memories, or caches on many-core on-chip systems. Performability evaluation of such a whole system is also another future research direction.

Acronyms

2D Two-Dimensional
3D Three-Dimensional
ANN Artificial Neural Network
BT Base Time
CCM Channel Connectivity Matrix
CMOS Complementary Metal Oxide Semiconductor
CTMC Continuous-Time Markov Chain
CT Communication Time
DOR Dimension-Ordered-Routing
DG Datasets Generator
DSP Digital Signal Processor
DTMC Discrete-Time Markov Chain
E East
FIFO First-In-First-Out
FTNF Fault-Tolerant Negative-First
HTTP Hypertext Transfer Protocol
I/O Input/Output
IP Intellectual Property
MFR Maximum Failure Rate
N North
NoC Network-on-Chip
NE North-East
NI Network Interface
NW North-West
PE Processing Element
QoS Quality of Service
RMSE Root Mean Squared Error
S South
SAF Store-and-Forward
SoC System-on-Chip
SE South-East
SW South-West
TSV Through-Silicon-Via
W West

References

[1] The official yaml web site. https://yaml.org/. Accessed: 2020-05-20

[2] Abdallah, A.B.: Advanced Multicore Systems-On-Chip. Springer (2017)

[3] Adve, V.S., Vernon, M.K.: Performance analysis of mesh interconnection networks with deterministic routing. IEEE Transactions on Parallel and Distributed Systems **5**(3), 225–246 (1994)

[4] Ahmed, A.B., Abdallah, A.B.: Graceful deadlock-free fault-tolerant routing algorithm for 3d network-on-chip architectures. Journal of Parallel and Distributed Computing **74**(4), 2229–2240 (2014)

[5] Al-Khalidi, H.R., Schnell, D.J.: Application of a continuous-time markov chain to a preclinical study. Drug information journal **31**(2), 607–613 (1997)

[6] Amer, H.H., Moustafa, M.S., Daoud, R.M.: Performability analysis for fault-tolerant wifi communication system. In: 2009 IEEE Conference on Emerging Technologies & Factory Automation, pp. 1–4. IEEE (2009)

[7] Anderson, D.F., Kurtz, T.G.: Continuous time markov chain models for chemical reaction networks. In: Design and analysis of biomolecular circuits, pp. 3–42. Springer (2011)

[8] Anghel, L., Nicolaidis, M.: Cost reduction and evaluation of a temporary faults-detecting technique. In: Design, Automation, and Test in Europe, pp. 423–438. Springer (2008)

[9] Arioli, M., Codenotti, B., Fassino, C.: The padé method for computing the matrix exponential. Linear algebra and its applications **240**, 111–130 (1996)

[10] Ascia, G., Catania, V., Palesi, M., Patti, D.: Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. IEEE Transactions on Computers **57**(6), 809–820 (2008)

[11] Awange, J.L., Paláncz, B., Lewis, R.H., Völgyesi, L.: Mathematical Geosciences. Springer (2018)

[12] Baccelli, F., Błaszczyszyn, B., et al.: Stochastic geometry and wireless networks: Volume ii applications. Foundations and Trends® in Networking **4**(1–2), 1–312 (2010)

[13] Bakhouya, M., Suboh, S., Gaber, J., El-Ghazawi, T.: Analytical performance comparison of 2d mesh, wk-recursive, and spidergon nocs. In: 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–6. IEEE (2010)

[14] Baron, M.: Probability and statistics for computer scientists. CRC Press (2019)

[15] Bekooij, M., Hoes, R., Moreira, O., Poplavko, P., Pastrnak, M., Mesman, B., Mol, J.D., Stuijk, S., Gheorghita, V., Van Meerbergen, J.: Dataflow analysis for real-time embedded multiprocessor system design. In: Dynamic and robust streaming in and between connected consumer-electronic devices, pp. 81–108. Springer (2005)

[16] Benini, L., De Micheli, G.: Networks on chips: A new soc paradigm. computer **35**(1), 70–78 (2002)

[17] Black, B., Annavaram, M., Brekelbaum, N., DeVale, J., Jiang, L., Loh, G.H., McCaule, D., Morrow, P., Nelson, D.W., Pantuso, D., et al.: Die stacking (3d) microar-

chitecture. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 469–479. IEEE Computer Society (2006)
[18] Bononi, L., Concer, N.: Simulation and analysis of network on chip architectures: ring, spidergon and 2d mesh. In: Proceedings of the Design Automation & Test in Europe Conference, vol. 2, pp. 6–pp. IEEE (2006)
[19] Borkar, S.: Thousand core chips: a technology perspective. In: Proceedings of the 44th annual design automation conference, pp. 746–749 (2007)
[20] Boutaba, R., Salahuddin, M.A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., Caicedo, O.M.: A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. Journal of Internet Services and Applications **9**(1), 16 (2018)
[21] Boyan, J.A., Littman, M.L.: Packet routing in dynamically changing networks: A reinforcement learning approach. In: Advances in neural information processing systems, pp. 671–678 (1994)
[22] Catania, V., Mineo, A., Monteleone, S., Palesi, M., Patti, D.: Noxim: An open, extensible and cycle-accurate network on chip simulator. In: 2015 IEEE 26th international conference on application-specific systems, architectures and processors (ASAP), pp. 162–163. IEEE (2015)
[23] Chan, W.H.R., Zhang, P., Nevat, I., Nagarajan, S.G., Valera, A.C., Tan, H.X., Gautam, N.: Adaptive duty cycling in sensor networks with energy harvesting using continuous-time markov chain and fluid models. IEEE Journal on Selected Areas in Communications **33**(12), 2687–2700 (2015)
[24] Chao Du, S.: Correlation analysis of enzymatic reaction of a single protein molecule. The annals of applied statistics **6**(3), 950 (2012)
[25] Chen, J., Li, C., Gillard, P.: Network-on-chip (noc) topologies and performance: a review. In: Proceedings of the 2011 Newfoundland Electrical and Computer Engineering Conference (NECEC), pp. 1–6 (2011)
[26] Chen, K.C., Ebrahimi, M., Wang, T.Y., Yang, Y.C.: Noc-based dnn accelerator: A future design paradigm. In: Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip, pp. 1–8 (2019)
[27] Cheng, A.l., Pan, Y., Yan, X.l., Huan, R.h.: A general communication performance evaluation model based on routing path decomposition. Journal of Zhejiang University SCIENCE C **12**(7), 561–573 (2011)
[28] Choi, S.P., Yeung, D.Y.: Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In: Advances in Neural Information Processing Systems, pp. 945–951 (1996)
[29] Cota, É., de Morais Amory, A., Lubaszewski, M.S.: Reliability, Availability and Serviceability of Networks-on-chip. Springer Science & Business Media (2011)
[30] Cover, T.M., Thomas, J.A.: Elements of information theory. John Wiley & Sons (2012)
[31] Dalirsani, A., Hosseinabady, M., Navabi, Z.: An analytical model for reliability evaluation of noc architectures. In: 13th IEEE International On-Line Testing Symposium (IOLTS 2007), pp. 49–56. IEEE (2007)
[32] Dally, W.J., Towles, B.: Route packets, not wires: on-chip inteconnection networks. In: Proceedings of the 38th annual Design Automation Conference, pp. 684–689 (2001)

[33] Dally, W.J., Towles, B.P.: Principles and practices of interconnection networks. Elsevier (2004)
[34] Dally, W.J., et al.: Virtual-channel flow control. IEEE Transactions on Parallel and Distributed systems **3**(2), 194–205 (1992)
[35] Dang, K.N., Ahmed, A.B., Tran, X.T., Okuyama, Y., Abdallah, A.B.: A comprehensive reliability assessment of fault-resilient network-on-chip using analytical model. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **25**(11), 3099–3112 (2017)
[36] Das, S., Doppa, J.R., Kim, D.H., Pande, P.P., Chakrabarty, K.: Optimizing 3d noc design for energy efficiency: A machine learning approach. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 705–712. IEEE Press (2015)
[37] DiTomaso, D., Kodi, A., Matolak, D., Kaya, S., Laha, S., Rayess, W.: A-winoc: Adaptive wireless network-on-chip architecture for chip multiprocessors. IEEE Transactions on Parallel and Distributed Systems **26**(12), 3289–3302 (2014)
[38] DiTomaso, D., Sikder, A., Kodi, A., Louri, A.: Machine learning enabled power-aware network-on-chip design. In: Proceedings of the Conference on Design, Automation & Test in Europe, pp. 1354–1359. European Design and Automation Association (2017)
[39] Dobrow, R.P.: Introduction to stochastic processes with R. John Wiley & Sons (2016)
[40] Dougherty, E.R.: Random processes for image and signal processing. SPIE press (1999)
[41] Duato, J., Yalamanchili, S., Ni, L.M.: Interconnection networks: an engineering approach. Morgan Kaufmann (2003)
[42] Dubois, F., Sheibanyrad, A., Petrot, F., Bahmani, M.: Elevator-first: A deadlock-free distributed routing algorithm for vertically partially connected 3d-nocs. IEEE Transactions on Computers **62**(3), 609–615 (2011)
[43] Duncan, A.: Powers of the adjacency matrix and the walk matrix (2004)
[44] Ebrahimi, M., Daneshtalab, M., Plosila, J.: Fault-tolerant routing algorithm for 3d noc using hamiltonian path strategy. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1601–1604. EDA Consortium (2013)
[45] Eggenberger, M., Radetzki, M.: Scalable parallel simulation of networks on chip. In: 2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS), pp. 1–8. IEEE (2013)
[46] Ejlali, A., Al-Hashimi, B.M., Rosinger, P., Miremadi, S.G., Benini, L.: Performability/energy tradeoff in error-control schemes for on-chip networks. IEEE transactions on very large scale integration (VLSI) systems **18**(1), 1–14 (2010)
[47] Elakkumanan, P., Prasad, K., Sridhar, R.: Time redundancy based scan flip-flop reuse to reduce ser of combinational logic. In: 7th International Symposium on Quality Electronic Design (ISQED'06), pp. 6–pp. IEEE (2006)
[48] Elmiligi, H., Morgan, A.A., El-Kharashi, M.W., Gebali, F.: Improving networks-on-chip performability: A topology-based approach. International Journal of Circuit Theory and Applications **39**(6), 557–572 (2011)
[49] Eusgeld, I., Happe, J., Limbourg, P., Rohr, M., Salfner, F.: Performability. In: Dependability metrics, pp. 245–254. Springer (2008)
[50] Feero, B.S., Pande, P.P.: Networks-on-chip in a three-dimensional environment: A performance evaluation. IEEE Transactions on computers **58**(1), 32–45 (2008)

[51] Flich, J., Bertozzi, D.: Designing network on-chip architectures in the nanoscale era. Chapman and Hall/CRC (2010)
[52] Florescu, I.: Probability and stochastic processes. John Wiley & Sons (2014)
[53] Gagniuc, P.A.: Markov chains: from theory to implementation and experimentation. John Wiley & Sons (2017)
[54] Gardea, J., Jin, Y., Badawy, A.H., Cook, J.: Performance evaluation of mesh-based 3d nocs. In: Proceedings of the 10th International Workshop on Network on Chip Architectures, pp. 1–6 (2017)
[55] Géron, A.: Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc." (2017)
[56] Ghosh, R., Trivedi, K.S., Naik, V.K., Kim, D.S.: End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In: 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, pp. 125–132. IEEE (2010)
[57] Glass, C.J., Ni, L.M.: The turn model for adaptive routing. ACM SIGARCH Computer Architecture News **20**(2), 278–287 (1992)
[58] Glass, C.J., Ni, L.M.: Fault-tolerant wormhole routing in meshes. In: Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on, pp. 240–249. IEEE (1993)
[59] Gregoire, M.: Professional C++. John Wiley & Sons (2014)
[60] Guerrier, P., Greiner, A.: A generic architecture for on-chip packet-switched interconnections. In: Design, Automation, and Test in Europe, pp. 111–123. Springer (2008)
[61] Guntupalli, L., Li, F.Y.: Dtmc modeling for performance evaluation of dw-mac in wireless sensor networks. In: 2016 IEEE Wireless Communications and Networking Conference, pp. 1–6. IEEE (2016)
[62] Gupta, A., Rawlings, J.B.: Comparison of parameter estimation methods in stochastic chemical kinetic models: examples in systems biology. AIChE Journal **60**(4), 1253–1268 (2014)
[63] Guz, Z., Walter, I., Bolotin, E., Cidon, I., Ginosar, R., Kolodny, A.: Network delays and link capacities in application-specific wormhole nocs. VLSI design **2007** (2007)
[64] Hajek, B.: Random processes for engineers. Cambridge university press (2015)
[65] Hall, B.: Lie groups, Lie algebras, and representations: an elementary introduction, vol. 222. Springer (2015)
[66] Hansson, A., Wiggers, M., Moonen, A., Goossens, K., Bekooij, M.: Applying dataflow analysis to dimension buffers for guaranteed performance in networks on chip. In: Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008), pp. 211–212. IEEE (2008)
[67] Hegedûs, A., Maggio, G.M., Kocarev, L.: A ns-2 simulator utilizing chaotic maps for network-on-chip traffic analysis. In: 2005 IEEE International Symposium on Circuits and Systems, pp. 3375–3378. IEEE (2005)
[68] Higham, N.J.: The scaling and squaring method for the matrix exponential revisited. SIAM Journal on Matrix Analysis and Applications **26**(4), 1179–1193 (2005)
[69] Hou, J., Han, Q., Radetzki, M.: A machine learning enabled long-term performance evaluation framework for nocs. In: 2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 164–171. IEEE (2019)

[70] Hou, J., Radetzki, M.: Performability analysis of mesh-based nocs using markov reward model. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 609–616. IEEE (2018)
[71] Hou, J., Radetzki, M.: A methodology to compute long-term fault resilience of nocs under fault-tolerant routing algorithms. In: 2019 Forum for Specification and Design Languages (FDL), pp. 1–7. IEEE (2019)
[72] Hou, K., Jia, H., Xu, X., Liu, Z., Jiang, Y.: A continuous time markov chain based sequential analytical approach for composite power system reliability assessment. IEEE Transactions on Power Systems **31**(1), 738–748 (2015)
[73] Hu, J., Ogras, U.Y., Marculescu, R.: System-level buffer allocation for application-specific networks-on-chip router design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **25**(12), 2919–2933 (2006)
[74] Hu, P.C., Kleinrock, L.: An analytical model for wormhole routing with finite size input buffers. In: Teletraffic Science and Engineering, vol. 2, pp. 549–560. Elsevier (1997)
[75] Jantsch, A., Sander, I.: Models of computation and languages for embedded system design. IEE Proceedings-Computers and Digital Techniques **152**(2), 114–129 (2005)
[76] Jerger, N.E., Peh, L.S.: On-chip networks. Synthesis Lectures on Computer Architecture **4**(1), 1–141 (2009)
[77] Jones, R.H., Xu, S., Grunwald, G.K.: Continuous time markov models for binary longitudinal data. Biometrical journal **48**(3), 411–419 (2006)
[78] Karlin, S.: A first course in stochastic processes. Academic press (2014)
[79] Ke, M., Li, C.: Analysis of sirs model on complex network based on the continuous time markov chain. In: 2013 International Conference on Computational and Information Sciences, pp. 1578–1581. IEEE (2013)
[80] Khichar, J., Choudhary, S., Mahar, R.: Fault tolerant dynamic xy-yx routing algorithm for network on-chip architecture. In: Intelligent Computing and Control (I2C2), 2017 International Conference on, pp. 1–6. IEEE (2017)
[81] Kim, J., Das, C., Lin, W., Feng, T.Y.: Reliability evaluation of hypercube multicomputers. IEEE Transactions on Reliability **38**(1), 121–129 (1989)
[82] Kim, J.H., Lee, S.M., Kim, D.S., Park, J.S.: Performability analysis of iaas cloud. In: 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp. 36–43. IEEE (2011)
[83] Koren, I., Krishna, C.M.: Fault-tolerant systems. Elsevier (2010)
[84] Kumar, S., Miikkulainen, R.: Dual reinforcement q-routing: An on-line adaptive routing algorithm. In: Proceedings of the artificial neural networks in engineering Conference, pp. 231–238 (1997)
[85] Le Boudec, J.Y., Thiran, P.: Network calculus: a theory of deterministic queuing systems for the internet, vol. 2050. Springer Science & Business Media (2001)
[86] Li, Q.K., Yang, N., Gao, W.W., Li, M.: Continuous-time markov chains based h∞ control of networked control systems. In: The 26th Chinese Control and Decision Conference (2014 CCDC), pp. 906–911. IEEE (2014)
[87] Losq, J., et al.: Effects of failures on gracefully degradable systems. (1977)
[88] Lu, Z.: Cross clock-domain tdm virtual circuits for networks on chips. In: Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip, pp. 209–216 (2011)

[89] Ma, J., Chan, W., Tilley, B.C.: Continuous time markov chain approaches for analyzing transtheoretical models of health behavioral change: a case study and comparison of model estimations. Statistical methods in medical research **27**(2), 593–607 (2018)
[90] Ma, Y., Han, J.J., Trivedi, K.S.: Composite performance and availability analysis of wireless communication networks. IEEE Transactions on Vehicular Technology **50**(5), 1216–1223 (2001)
[91] Mahgoub, I.O.: Reliability of cluster-based multiprocessor systems. In: 1992 12th International Conference on Distributed Computing System, pp. 204–205. IEEE Computer Society (1992)
[92] Meyer, C.D.: Matrix analysis and applied linear algebra, vol. 71. Siam (2000)
[93] Meyer, J.F.: Models and techniques for evaluating the effectiveness of aircraft computing systems (1978)
[94] Meyer, J.F.: On evaluating the performability of degradable computing systems. IEEE Transactions on computers (8), 720–731 (1980)
[95] Mhoon, K.B., Chan, W., Del Junco, D.J., Vernon, S.W.: A continuous-time markov chain approach analyzing the stages of change construct from a health promotion intervention. JP journal of biostatistics **4**(3), 213 (2010)
[96] Mijumbi, R., Gorricho, J.L., Serrat, J., Claeys, M., De Turck, F., Latré, S.: Design and evaluation of learning algorithms for dynamic resource management in virtual networks. In: 2014 IEEE network operations and management symposium (NOMS), pp. 1–9. IEEE (2014)
[97] Mo, J.: Performance modeling of communication networks with Markov chains. Morgan & Claypool Publishers (2010)
[98] Moadeli, M., Shahrabi, A., Vanderbauwhede, W., Ould-Khaoua, M.: An analytical performance model for the spidergon noc. In: 21st International Conference on Advanced Information Networking and Applications (AINA'07), pp. 1014–1021. IEEE (2007)
[99] Mondal, M., Wu, X., Aziz, A., Massoud, Y.: Reliability analysis for on-chip networks under rc interconnect delay variation. In: 2006 1st International Conference on Nano-Networks and Workshops, pp. 1–5. IEEE (2006)
[100] Moriam, S., Fettweis, G.P.: Fault tolerant deadlock-free adaptive routing algorithms for hexagonal networks-on-chip. In: Digital System Design (DSD), 2016 Euromicro Conference on, pp. 131–137. IEEE (2016)
[101] Moriam, S., Fettweis, G.P.: Reliability assessment of fault tolerant routing algorithms in networks-on-chip: An analytic approach. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 61–66. IEEE (2017)
[102] Newman, M.: Networks. Oxford university press (2018)
[103] Ni, L.M., McKinley, P.K.: A survey of wormhole routing techniques in direct networks. Computer **26**(2), 62–76 (1993)
[104] Nicolaidis, M.: Time redundancy based soft-error tolerance to rescue nanometer technologies. In: Proceedings 17th IEEE VLSI Test Symposium (Cat. No. PR00146), pp. 86–94. IEEE (1999)
[105] Ogras, U.Y., Bogdan, P., Marculescu, R.: An analytical approach for network-on-chip performance analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **29**(12), 2001–2013 (2010)
[106] Ogras, U.Y., Marculescu, R.: Analytical router modeling for networks-on-chip performance analysis. In: 2007 Design, Automation & Test in Europe Conference & Exhibition, pp. 1–6. IEEE (2007)

[107] Okamura, H., Zheng, J., Dohi, T.: A statistical framework on software aging modeling with continuous-time hidden markov model. In: 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS), pp. 114–123. IEEE (2017)
[108] Panda, P.R.: Systemc-a modeling platform supporting multiple design abstractions. In: International Symposium on System Synthesis (IEEE Cat. No. 01EX526), pp. 75–80. IEEE (2001)
[109] Pande, P.P., Grecu, C., Jones, M., Ivanov, A., Saleh, R.: Performance evaluation and design trade-offs for network-on-chip interconnect architectures. IEEE transactions on Computers **54**(8), 1025–1040 (2005)
[110] Pavlidis, V.F., Friedman, E.G.: 3-d topologies for networks-on-chip. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **15**(10), 1081–1090 (2007)
[111] Peh, L.S., Dally, W.J.: Flit-reservation flow control. In: Proceedings Sixth International Symposium on High-Performance Computer Architecture. HPCA-6 (Cat. No. PR00550), pp. 73–84. IEEE (2000)
[112] Pulungan, R., Hermanns, H.: Transient analysis of ctmcs: Uniformization or matrix exponential? IAENG International Journal of Computer Science **45**(2), 267–274 (2018)
[113] Qian, Y., Lu, Z., Dou, W.: Analysis of communication delay bounds for network on chips. In: 2009 Asia and South Pacific Design Automation Conference, pp. 7–12. IEEE (2009)
[114] Qian, Y., Lu, Z., Dou, W.: Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In: 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip, pp. 44–53. IEEE (2009)
[115] Qian, Y., Lu, Z., Dou, W.: From 2d to 3d nocs: a case study on worst-case communication performance. In: 2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers, pp. 555–562. IEEE (2009)
[116] Qian, Z., Juan, D.C., Bogdan, P., Tsui, C.Y., Marculescu, D., Marculescu, R.: Svr-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model. In: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 354–357. IEEE (2013)
[117] Radetzki, M., Feng, C., Zhao, X., Jantsch, A.: Methods for fault tolerance in networks-on-chip. ACM Computing Surveys (CSUR) **46**(1), 8 (2013)
[118] Rahmani, A.M., Latif, K., Liljeberg, P., Plosila, J., Tenhunen, H.: Research and practices on 3d networks-on-chip architectures. In: NORCHIP 2010, pp. 1–6. IEEE (2010)
[119] Refan, F., Alemzadeh, H., Safari, S., Prinetto, P., Navabi, Z.: Reliability in application specific mesh-based noc architectures. In: On-Line Testing Symposium, 2008. IOLTS'08. 14th IEEE International, pp. 207–212. IEEE (2008)
[120] Reibman, A., Trivedi, K.: Numerical transient analysis of markov models. Computers & Operations Research **15**(1), 19–36 (1988)
[121] Ren, P., Lis, M., Cho, M.H., Shim, K.S., Fletcher, C.W., Khan, O., Zheng, N., Devadas, S.: Hornet: A cycle-level multicore simulator. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **31**(6), 890–903 (2012)
[122] Ross, S.M., Kelly, J.J., Sullivan, R.J., Perry, W.J., Mercer, D., Davis, R.M., Washburn, T.D., Sager, E.V., Boyce, J.B., Bristow, V.L.: Stochastic processes, vol. 2. Wiley New York (1996)

[123] Safaei, F., Khonsari, A., Fathy, M., Ould-Khaoua, M.: Performance analysis of fault-tolerant routing algorithm in wormhole-switched interconnections. The Journal of Supercomputing **41**(3), 215–245 (2007)
[124] Sahner, R.A., Trivedi, K., Puliafito, A.: Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Springer Science & Business Media (2012)
[125] Salamat, R., Khayambashi, M., Ebrahimi, M., Bagherzadeh, N.: A resilient routing algorithm with formal reliability analysis for partially connected 3d-nocs. IEEE Transactions on Computers **65**(11), 3265–3279 (2016)
[126] Salamat, R., Khayambashi, M., Ebrahimi, M., Bagherzadeh, N.: Lead: An adaptive 3d-noc routing algorithm with queuing-theory based analytical verification. IEEE Transactions on Computers **67**(8), 1153–1166 (2018)
[127] Salem, A.A., El Ghany, M.A.A., Hofmann, K.: Performability measurement of coding algorithms for network on chip. In: Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on, pp. 691–694. IEEE (2013)
[128] Sass, J., Haussmann, U.G.: Optimizing the terminal wealth under partial information: The drift process as a continuous time markov chain. Finance and Stochastics **8**(4), 553–577 (2004)
[129] e Silva, E.d.S., Gail, H.R.: Performability analysis of computer systems: from model specification to solution. Performance Evaluation **14**(3), 157–196 (1992)
[130] Simeone, O.: A very brief introduction to machine learning with applications to communication systems. IEEE Transactions on Cognitive Communications and Networking **4**(4), 648–664 (2018)
[131] Smith, R., Trivedi, K.S., Ramesh, A.: Performability analysis: measures, an algorithm, and a case study. IEEE Transactions on Computers **37**(4), 406–417 (1988)
[132] Stuijk, S., Basten, T., Geilen, M., Corporaal, H.: Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In: 2007 44th ACM/IEEE Design Automation Conference, pp. 777–782. IEEE (2007)
[133] Su, G., Chen, T., Feng, Y., Rosenblum, D.S.: Proeva: runtime proactive performance evaluation based on continuous-time markov chains. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 484–495. IEEE (2017)
[134] Sun, H., Wang, X., Wang, C., Wang, R.: Performance analysis of network attack based on continuous time markov chain in dynamic spectrum access networks. In: 2017 9th International Conference on Modelling, Identification and Control (ICMIC), pp. 302–307. IEEE (2017)
[135] Sun, Y., Yin, X., Jiang, J., Sekar, V., Lin, F., Wang, N., Liu, T., Sinopoli, B.: Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In: Proceedings of the 2016 ACM SIGCOMM Conference, pp. 272–285 (2016)
[136] Sun, Y.R., Kumar, S., Jantsch, A.: Simulation and evaluation for a network on chip architecture using ns-2. In: Proceedings of the IEEE NorChip conference, pp. 167–172. Citeseer (2002)
[137] Tang, M., Lin, X.: Injection level flow control for networks-on-chip (noc). J. Inf. Sci. Eng. **27**(2), 527–544 (2011)
[138] Tong, H., Brown, T.X.: Adaptive call admission control under quality of service constraints: a reinforcement learning solution. IEEE Journal on selected Areas in Communications **18**(2), 209–221 (2000)

[139] Topol, A.W., La Tulipe, D., Shi, L., Frank, D.J., Bernstein, K., Steen, S.E., Kumar, A., Singco, G.U., Young, A.M., Guarini, K.W., et al.: Three-dimensional integrated circuits. IBM Journal of Research and Development **50**(4.5), 491–506 (2006)
[140] Trivedi, K.S.: Probability and statistics with reliability, queuing, and computer science applications, vol. 13. Wiley Online Library (1982)
[141] Trivedi, K.S., Andrade, E.C., Machida, F.: Combining performance and availability analysis in practice. Advances in Computers **84**, 1–38 (2012)
[142] Valiant, L.G.: A scheme for fast parallel communication. SIAM journal on computing **11**(2), 350–361 (1982)
[143] Valinataj, M., Mohammadi, S., Safari, S.: Reliability assessment of networks-on-chip based on analytical models. Journal of Zhejiang University-SCIENCE A **10**(12), 1801–1814 (2009)
[144] Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., et al.: An 80-tile 1.28 tflops network-on-chip in 65nm cmos. In: 2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, pp. 98–589. IEEE (2007)
[145] Wang, G., Wang, G., Chen, S., et al.: Probability model for faults in large-scale multicomputer systems. In: 2003 Test Symposium, pp. 452–457. IEEE (2003)
[146] Wang, J., Qiu, Y.: A new call admission control strategyfor lte femtocell networks. In: 2nd International Conference on Advances in Computer Science and Engineering (CSE 2013). Atlantis Press (2013)
[147] Wang, L.T., Stroud, C.E., Touba, N.A.: System-on-chip test architectures: nanometer design for testability. Morgan Kaufmann (2010)
[148] Wang, M., Cui, Y., Wang, X., Xiao, S., Jiang, J.: Machine learning for networking: Workflow, advances and opportunities. IEEE Network **32**(2), 92–99 (2017)
[149] Weerasekera, R., Zheng, L.R., Pamunuwa, D., Tenhunen, H.: Extending systems-on-chip to the third dimension: performance, cost and technological tradeoffs. In: Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design, pp. 212–219. IEEE Press (2007)
[150] West, D.B., et al.: Introduction to graph theory, vol. 2. Prentice hall Upper Saddle River, NJ (1996)
[151] Winstein, K., Balakrishnan, H.: Tcp ex machina: Computer-generated congestion control. ACM SIGCOMM Computer Communication Review **43**(4), 123–134 (2013)
[152] worm_sim: Cycle accurate noc simulator. `https://research.ece.cmu.edu/sld/software/` (2005). [Online; accessed 20-June-2020]
[153] Wunderlich, H.J., Radetzki, M.: Multi-layer test and diagnosis for dependable nocs. In: Proceedings of the 9th International Symposium on Networks-on-Chip, pp. 1–8 (2015)
[154] Xu, J., Wolf, W., Henkel, J., Chakradhar, S.: A design methodology for application-specific networks-on-chip. ACM Transactions on Embedded Computing Systems (TECS) **5**(2), 263–280 (2006)
[155] Xu, L., Zhang, W., Chen, L.: Modeling users' visiting behaviors for web load testing by continuous time markov chain. In: 2010 Seventh Web Information Systems and Applications Conference, pp. 59–64. IEEE (2010)
[156] Yaghini, P.M., Eghbal, A., Pedram, H., Zarandi, H.R.: Investigation of transient fault effects in synchronous and asynchronous network on chip router. Journal of Systems Architecture **57**(1), 61–68 (2011)

[157] Zamzam, D.M., El Ghany, M.A.A., Hofmann, K.: Performability of error control schemes for noc interconnects. In: NORCHIP, 2012, pp. 1–5. IEEE (2012)
[158] Zhang, D., Zhang, X.: Study on forecasting the stock market trend based on stochastic analysis method. International Journal of Business and Management **4**(6), 163–170 (2009)
[159] Zhu, M., Lee, J., Choi, K.: An adaptive routing algorithm for 3d mesh noc with limited vertical bandwidth. In: 2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC), pp. 18–23. IEEE (2012)
[160] Zsigmond, G., Homolya, S., Lendvay, M.: Application of continuous-time markov chains by reliability analysis. In: 2009 7th International Symposium on Intelligent Systems and Informatics, pp. 69–72. IEEE (2009)